MASTER EN SCIENCES INFORMATIQUES

Réseaux neuronaux pour la reconnaissance de formulaires

Bekaert, Koenraad

Award date:
1992

FACULTES
UNIVERSITAIRES
NOTRE DAME DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

RESEAUX NEURONAUX POUR LA RECONNAISSANCE DE FORMULAIRES.

Mémoire de licence et maîtrise en informatique.

Koenraad BEKAERT

Promoteur: Le Père Recteur J. BERLEUR s.j.

Co-Promoteur: Monsieur le Professeur J.-P. PETERS

Année Académique 1991 - 1992

RUE GRANDGAGNAGE, 21, B - 5000 NAMUR (BELGIUM)

Résumé.

La reconnaissance de formulaires a été étudiée et implémentée comme partie d'un système automatique d'analyse de formulaires. Nous nous attendons à ce que ce sous-système fonctionne de façon optimale en la présence de rotation, translation et modification d'échelle du formulaire.

La faisabilité d'invariance à l'égard de ces transformations géométriques a été testée avec différents types de réseaux neuronaux. Pour les types de réseaux connus comme le Perceptron à plusieurs couches et le Hopfield, ainsi que pour le plus complexe réseau neuronal d'Ordre Supérieur, des réponses concluantes sont présentées.

Une implémentation non-neuronale a également été réalisée et est présentée comme méthode classique, en opposition avec les réseaux neuronaux innovateurs.

Abstract.

As a part of an automatic forms analysis system, recognition of forms has been studied and implemented. We expect this subsystem to perform optimally in the presence of rotation, translation and scaling of the forms.

The feasibility of invariance in the face of these geometric transformations has been tested with various kinds of neuronal networks. Conclusive answers are presented herein for the well-known multilayer Perceptron and Hopfield type neuronal nets and the more complex Higher Order neuronal networks.

An implementation has also been realized in a non-neural way and is presented here as the classical method, as opposed to the innovative neural nets.

Remerciements.

Je tiens à exprimer ma vive reconnaissance envers tous ceux qui m'ont assisté et guidé par leurs conseils.

Ma profonde gratitude s'adresse tout particulièrement:

Au Père Recteur J. Berleur s.j., mon promoteur, qui m'a dirigé vers la recherche dans le domaine des réseaux neuronaux et dont les encouragements m'ont été d'une aide précieuse.

A Monsieur J.P. Peters, mon co-promoteur, pour les conseils judicieux qu'il m'a prodigué et pour ma prise en charge, patiente et bienveillante.

A Monsieur Bart De Greef, chercheur chez Philips, car sans son appui incessant mon stage à Eindhoven n'aurait pû être aussi fructueux.

Je porte également une profonde reconnaissance aux professeurs de l'institut ainsi qu'à mes parents qui, par leur compréhension et leur soutient infaillible, m'ont permis de mener à bien mes études et d'aboutir à l'élaboration de ce mémoire.

Je ne voudrais terminer ces remerciements sans citer l'assistance impromptue d'une co-stagiaire de Philips, Ana Peris.

INTRODUCTION

De tout domaine inconnu émane une attraction. Les réseaux neuronaux[1] constituent un nouveau monde à explorer. Quelques propriétés des réseaux neuronaux, comme par exemple l'abstraction, la généralisation ou l'apprentissage, sont porteurs de grandes richesses potentielles.

Après quelques décennies d'oubli, les recherches en réseaux neuronaux reprennent leur élan. La recherche empirique est plus importante que la recherche théorique. Du moins, les résultats pratiques sont plus nombreux que les résultats théoriques. Dans ce mémoire nous contribuons à la recherche de solutions pratiques à un problème bien concret, celui de la reconnaissance de formulaires.

Dans le premier chapitre nous développerons la reconnaissance de formulaires. Ceci situera le problème à résoudre.

Le deuxième chapitre nous guidera à travers une introduction des réseaux neuronaux et la théorie des réseaux neuronaux utilisés pour la reconnaissance des formulaires.

Dans le chapitre trois, deux applications sont décrites en détail. Elles utilisent des implémentations existantes. Il s'agit du "feature matching" réalisé par un réseau neuronal du type Perceptron et par un réseau neuronal du type Hopfield.

Le quatrième chapitre détaille une réalisation du "pattern matching" avec un réseau neuronal d'Ordre Supérieur.

Le chapitre cinq décrit un algorithme non-neuronal, écrit dans le cadre du stage, résolvant la reconnaissance de formulaires. Cette partie permet d'évaluer l'apport de la méthode neuronale.

---

[1] Tout au long du mémoire il s'agit de réseaux neuronaux **formels**, qui n'ont rien ou peu en commun avec leurs contreparties biologiques.

Nous concluons en établissant une comparaison entre les résultats des quatre implémentations et nous proposons également quelques recherches ultérieures.

Une postface est consacrée à quelques réflexions sur l'applicabilité des réseaux neuronaux et leur place dans la cognition et les sciences cognitives.

L'annexe A reprend le vocabulaire propre au sujet.

L'annexe B est formée par le détail algorithmique des applications.

La bibliographie se trouve en l'annexe C.

RECONNAISSANCE DE FORMULAIRES

I.1. Un système de traitement de formulaires.

Chacun de nous a été confronté maintes fois à des formulaires. Ce sont des documents contenant du texte pré-imprimé indiquant quelles données remplir dans des espaces blancs bien déterminés. Pour en faciliter l'usage, les zones libres sont souvent encadrées.

Ces formulaires sont ensuite encodés sur ordinateur pour subir des traitements automatisés. Cet encodage est dans la plupart des cas fait par des personnes. Il s'agit d'un travail lent et fastidieux et ouvert à l'erreur. Une entreprise se doit d'éviter au maximum de telles erreurs. Une ouverture existe donc sur le marché pour des systèmes automatiques de reconnaissance, voir d'analyse de formulaires.

Pour automatiser dans la mesure du possible l'étape d'encodage des formulaires, un système d'analyse de formulaires a été réalisé chez Philips. Il est divisé en quatre grandes phases :

- la digitalisation du formulaire,
- la reconnaissance du formulaire,
- l'extraction des données remplies par l'utilisateur et
- la reconnaissance des données extraites.

La phase de **digitalisation du formulaire** ne nécessite pas de développement. Le formulaire passe à travers un scanner et est sauvé sous forme d'image digitale en mémoire. Une seconde possibilité: un formulaire est reçu par fax et est envoyé, sous sa forme digitale, vers le système d'analyse qui le sauve en mémoire.

La **reconnaissance du formulaire** se base sur les lignes horizontales et verticales du formulaire. Nous recherchons le nom du formulaire, indépendamment de toute transformation géométrique. Cette phase de reconnaissance constitue le sujet de ce mémoire et sera traité plus en profondeur dans les chapitres qui suivent.

La phase suivante concerne l'**extraction des données remplies** sur le formulaire. Dit de façon simple, il suffit de "retirer" l'information pré-imprimée sur le formulaire pour obtenir les données remplies. Une base de données contient tous les formulaires vierges que le système doit pouvoir analyser. En prenant le formulaire vierge correspondant au formulaire rempli et en faisant correspondre parfaitement, par des transformations géométriques, ces deux images digitales, il est possible d'extraire les données remplies. Une implémentation de ce principe peut être décrite dans [Philips-2, 1990].

Dans les zones où de l'information alpha-numérique est attendue, on peut procéder à la **reconnaissance de caractères** par OCR (Optical Character Recognizer). Celui-ci traduit des caractères dans une image digitale en caractères codés binairement, par exemple en ASCII. Il fournit, avec la traduction, une valeur de confiance, et l'application décide alors d'accepter ou non la traduction.

Le résultat de cette analyse est un formulaire électronique correspondant au formulaire en papier. Si toute l'analyse s'est bien passée, le document électronique contient les caractères alpha-numériques traduits et éventuellement des images digitales correspondant aux données graphiques remplies sur le formulaire. Tant pendant la reconnaissance du formulaire que pendant la reconnaissance des caractères, l'aide d'un opérateur externe peut être demandée si un problème de reconnaissance ou de traduction se manifeste.

I.2. La phase de reconnaissance du formulaire.

I.2.1 SPÉCIFICATIONS.

Ce mémoire est axé sur la reconnaissance de formulaires donnés, en présence de transformations géométriques. Nous nous penchons sur la possibilité, ou l'impossibilité, d'une telle reconnaissance, principalement par quelques types de réseaux neuronaux, mais également par une méthode classique. En corollaire, et uniquement dans le dernier cas, nous déterminons la valeur des paramètres de transformation.

En **entrée**, nous retrouvons
1. une *image digitalisée* du formulaire à reconnaître, et
2. une *base de données de formulaires vierges* que le système doit pouvoir traiter.

En **sortie**, la phase d'extraction de données requiert
1. le *nom du formulaire* repris dans la base de données, et
2. la *transformation géométrique* à appliquer pour faire correspondre le formulaire rempli avec le formulaire vierge de la base de données.

I.2.2. HYPOTHÈSES ET CONTRAINTES DE TRAVAIL.

1. Le système de traitement est prévu pour des documents courants, basés sur des formulaires. La grande majorité des formulaires contient des lignes horizontales et/ou verticales et des rectangles formés par des lignes horizontales et verticales[2].

2. Il est également acceptable de supposer que les formulaires à traiter sont près du format standard A4. Et en conséquence de limiter l'échelle à 70% (le format A5) vers le bas et 140% (le format A3) vers le haut. La nécessité de cette contrainte sera expliquée ultérieurement.

3. Comme nous nous trouvons dans un environnement de traitement professionnel, on demande que la phase d'analyse soit robuste et rapide, surtout dans les cas de routine. Par exemple, on exige pour un formulaire connu par le système, rempli avec des caractères imprimés et qui n'a pas changé d'échelle ou été pivoté, d'être reconnu rapidement et correctement.

4. Les cas plus exceptionnels mais qui doivent également être reconnus, sont des rotations du formulaire de multiples de 90 degrés.

I.2.3. UN PROCESSUS EN DEUX ÉTAPES.

La phase de reconnaissance est divisée en deux étapes très distinctes (voir figure 1-1.) Dans une première étape, le formulaire digitalisé est pré-traité. Ce **pré-traitement** est une sorte d'opération de normalisation du formulaire, qui facilite considérablement la reconnaissance. Ce pré-traitement est nécessaire autant pour la reconnaissance par les réseaux neuronaux que par la méthode classique.

Après cette étape, la forme normalisée d'un formulaire vierge peut servir de modèle. Dans ce cas, cette forme normalisée doit être apprise

---

[2] Dans le cadre du travail que nous avons réalisé chez Philips, l'utilisation de lignes a été imposée.

par le réseau neuronal; ou ajouté à la base de données, dans le cas de la méthode classique.

La deuxième étape est la **reconnaissance** proprement dite du formulaire et le calcul des paramètres de transformation. Nous reviendrons sur cette dernière étape dans le chapitre V.

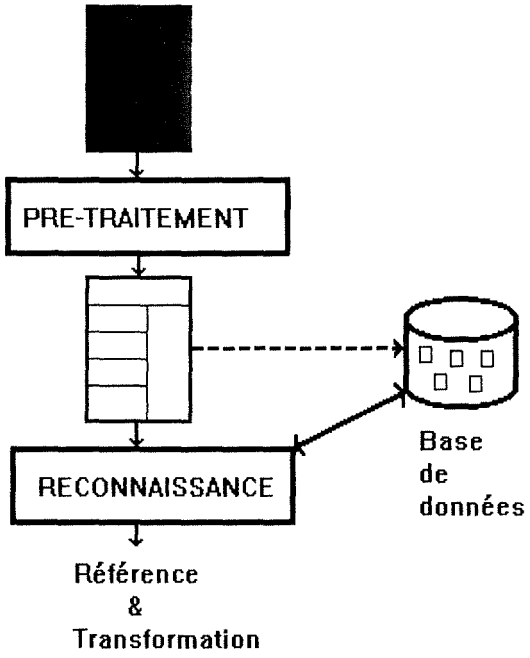

Figure 1-1 : La phase de reconnaissance.

I.2.4. LE PRÉ-TRAITEMENT.

Le pré-traitement est divisé en quatre parties. Vient d'abord l'extraction des lignes du formulaire. Basé sur l'orientation des lignes, nous redressons le formulaire. Ensuite nous appliquons des simplifications aux lignes et aux intersections de lignes. En dernier lieu, nous extrayons les caractéristiques du formulaire.

I.2.4.1. Extraction des lignes.

A partir de l'image digitalisé du formulaire, les lignes horizontales, verticales et légèrement obliques sont extraites. Par souci d'espace mémoire, elles sont sauvées sous forme analytique à partir des coordonnées des points extrêmes.

Au terme, nous obtenons une espèce de squelette du formulaire, que nous appelerons dorénavant : **un frame**.

Plus formellement :

Argument : Un bitmap.
Précondition :
Le bitmap est une image binaire d'un formulaire contenant des lignes horizontales et/ou verticales.

Résultats : Le frame correspondant au formulaire, contenant :
1) une liste de descriptions de lignes :
a) les coordonnées x et y des points extrémaux des lignes,
b) l' angle de la ligne, et
c) la longueur de la ligne,
2) longueur et largeur du bitmap.
Postcondition :
Si une ligne est décrite par ces points extrémes a et b, dont les coordonnées sont respectivement (ax, ay) et (bx, by),
alors

a) la longueur est calculée par $\sqrt{(bx - ax)^2 + (by - ay)^2}$,

b) l'angle de la ligne est calculée par $ATAN(\frac{by - ay}{bx - ax})$

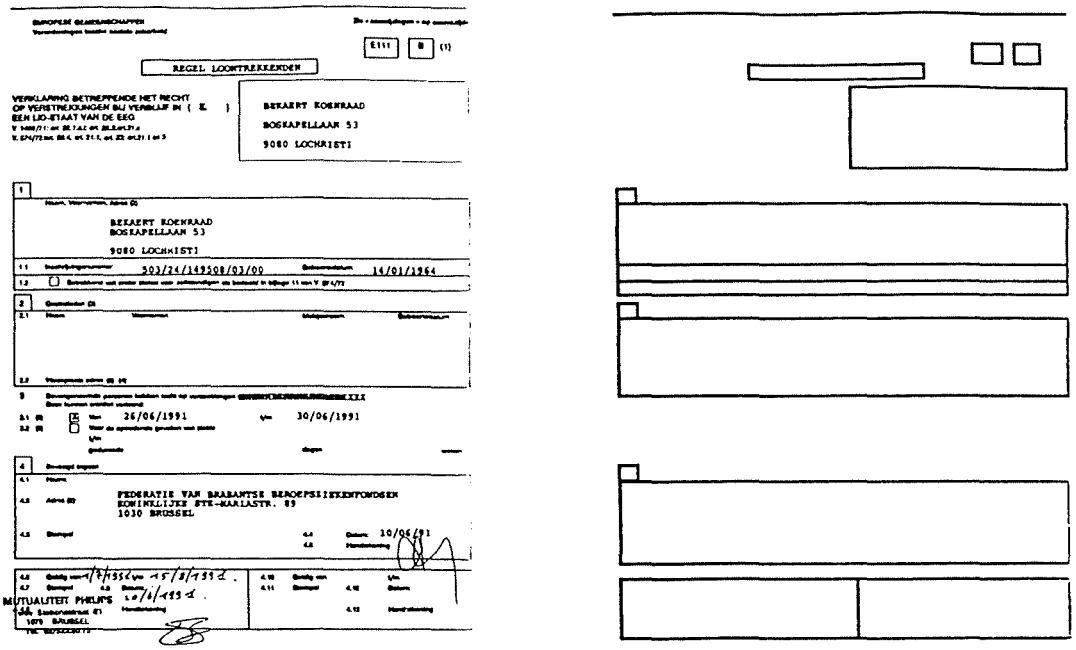

Figure 1-2 : Le formulaire E111 et son frame correspondant.

I.2.4.2. Orientation du frame.

Nous avons supposé qu'un formulaire contient en majorité des lignes horizontales et verticales (voir I.2.2.). En examinant l'angle des lignes que nous avons extraites, il est possible de rectifier, par rotation autour du centre de la page, l'orientation du frame. Cet ajustement fait partie des paramètres de transformation à transmettre en sortie. En même temps nous vérifions si l'orientation rencontrée n'excède pas les limites de l'OCR. Cette opération simplifie également d'une façon considérable la partie suivante.

Plus formellement:

Arguments :

frame F,
angle de rotation α.

Préconditions :

L'angle $\alpha \leq$ angle maximal de reconnaissance de caractères par l'OCR.

Résultats :

frame F'

Postconditions :

1) F'=F après rotation de α par la formule

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} xc \\ yc \end{bmatrix}$$

où x' et y' sont les coordonnées après rotation
et x et y sont les coordonnées avant rotation
et xc et yc sont les coordonnées du centre du formulaire.

2) La majorité des lignes sont horizontales et/ou verticales.

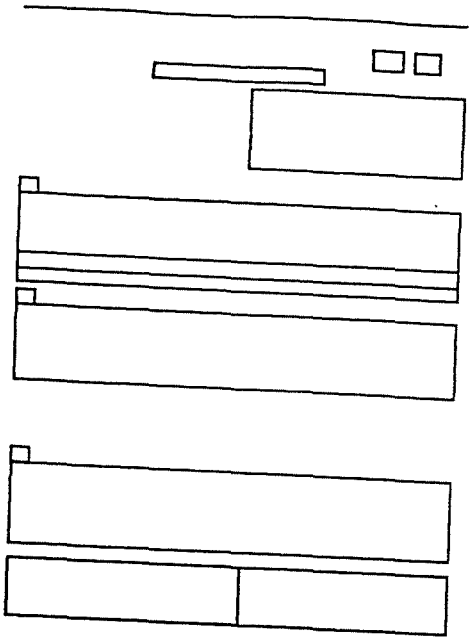

Figure 1-3 : Le frame du formulaire E111 pivoté de quelques degrés.

I.2.4.3. Simplification du frame.

Un formulaire digitalisé peut être "bruité" par plusieurs causes : le formulaire est mal rempli; l'état physique du formulaire en papier est tel que le scanner interprète le bruit comme des lignes ou des taches; ou bien le scanner lui-même ajoute du bruit. L'extraction des lignes ajoute également des impuretés. Dans cette phase de simplification du frame, nous essayons d'éliminer au maximum ces bruits et impuretés.

Une première simplification concerne les lignes. Nous réduisons le nombre de lignes de deux façons:

- deux lignes colinéaires proches l'une de l'autre sont jointes pour ne plus former qu'une seule ligne (voir figure 1-4) et
- deux lignes parallèles proches sont fusionnées en une seule (voir figure 1-5).

Figure 1-4 : Avant simplification après simplification.

Figure 1-5 : Avant simplification après simplification.

Dans un deuxième temps, les intersections des lignes sont purifiées. Nous ne nous intéressons qu'aux intersections de type "T", "L" ou "+". Dans la partie I.2.4.4., ce choix est justifié. Ce nettoyage à un niveau plus microscopique consiste à rendre les intersections exactes (voir figure 1-6.) Les traitements par ordinateur demandent de la précision. Par exemple, si la ligne d'une intersection du type "T" déborde, elle est réduite pour former une intersection "T" parfaite.

Figure 1-6 : Simplifications des intersections.

Dès lors nous parlons de **frame simplifié**. Le frame simplifié est fait en fonction des caractéristiques utilisées pour la reconnaissance du formulaire.

Plus formellement:

Arguments :
1) frame F
2) distance D

Précondition :
1) La majorité des lignes du frame F sont horizontales et/ou verticales.
2) La valeur de D a été trouvé expérimentalement et est propre au système de Philips.

Résultat :
Frame F'

Postconditions :
Frame F' = frame F, avec
1) toutes les lignes parallèles séparées d'au moins une distance D,
2) toutes les intersections des horizontales et verticales sont "propres":

Si l'horizontale H = H1 + H2 (segments se trouvant de chaque côté de la verticale. Voir figure 1-7.) et
si la verticale V = V1 + V2,
alors on doit avoir :

$$len(V1) = 0 \vee len(V1) > D \wedge$$
$$len(V2) = 0 \vee len(V2) > D \wedge$$
$$len(H1) = 0 \vee len(H1) > D \wedge$$
$$len(H2) = 0 \vee len(H2) > D.$$

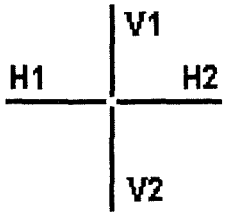

Figure 1-7

I.2.4.4. Extraction des caractéristiques.

Pour le choix des caractéristiques nous pouvons nous baser sur la représentation du formulaire ou sur sa structure.

Cette dernière demande une découpe du formulaire en ses composantes logiques; par exemple rectangles et lignes avec leurs attributs (coordonnées, ...) Elle est alors sauvée sous forme d'arbre dans la base de données. La reconnaissance se fait par recherche de correspondance entre deux graphes.

Les caractéristiques extraites à partir de la représentation du formulaire sont dans la majorité des cas plus simples, plus rapides à calculer et plus faciles à gérer. Nous optons pour cette dernière méthode.

Il faut faire un choix entre une multitude de possibilités, mais les caractéristiques doivent être basées uniquement sur les lignes horizontales et verticales apparaissant sur les formulaires. Elles doivent être indépendantes de toute échelle (E), de toute translation (T) et de toute rotation (R). En même temps, les caractéristiques doivent être suffisamment robustes pour admettre du bruit (B) sur le formulaire.

Examinons quelques caractéristiques par rapport aux critères donnés. Un "+" indique une invariance par rapport à cette transformation.

Caractéristiques		T	R	E	B
Nombre de lignes	:	+	+	+	(+)
Longueur des lignes	:	+	+		
Nombre et types de rectangles	:	+	+	+	
Surface et endroit des rectangles	:				
Ratio rectangles/lignes	:	+	+	+	
Nombre d'intersections de lignes	:	+	+	+	(+)

Une combinaison du nombre d'intersections, de lignes horizontales et verticales, et des nombres de lignes horizontales et verticales, est choisi comme critère de distinction entre frames. Les intersections ou les nombres de lignes se sont avérés, par expérimentation, insuffisants. Comme il s'agit de lignes horizontales et verticales uniquement, les intersections sont de trois types différents ("T", "L" et "+") mais, dans les quatre orientations possibles pour les deux premiers types. Nous comptons donc neuf types d'intersections.

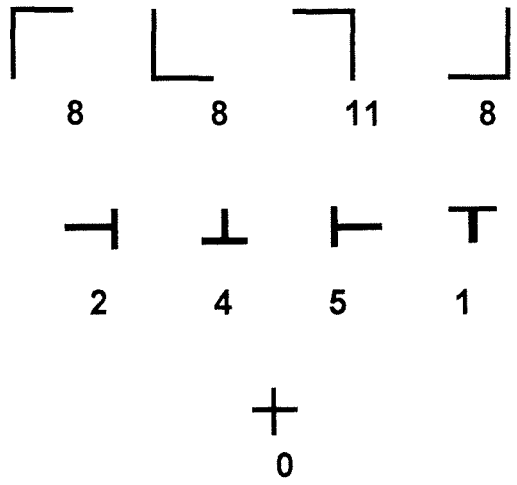

Figure 1-8 : Le nombre d'intersection de chaque type pour le formulaire E111.

Plus formellement :

Argument : frame F
Précondition : F est un frame simplifié.

Résultats : F' : frame simplifié avec caractéristiques.
Postconditions : F' = F + une liste contenant le nombre de
1) intersections de type "T" dans les 4 directions,
2) intersections de type "L" dans les 4 directions,
3) intersections de type "+",
4) lignes horizontales,
5) lignes verticales.

Rappelons qu'à ce stade le frame d'un formulaire vierge peut être inséré dans la base de données, pour la méthode classique, ou être appris par le réseau comme référence.

Dans ce chapitre nous avons présenté le problème: la reconnaissance de formulaires. Pour atteindre cet objectif nous nous servons soit des lignes de l'image de départ sous forme de bitmap, soit de la structure élémentaire formée par les lignes du formulaire et des caractéristiques que nous pouvons en déduire.

Dans mon travail chez Philips j'ai utilisé une méthode classique pour la reconnaissance (cf. chapitre V.) Dans ce mémoire nous nous tournons vers un domaine moins exploré et maîtrisé : les réseaux neuronaux.

Notre cerveau permet de reconnaître un visage, une personne, une mélodie, même une odeur ou une forme avec une aisance étonnante compte tenu de la diversité du monde extérieur. Cette aisance est nullement présente dans la reconnaissance par des méthodes classiques. Il faut donc chercher dans une autre direction : essayer de se rapprocher du fonctionnement du cerveau, du système neuronal.

Tenter de simuler le réseau neuronal biologique n'est pas nouveau. Un grand nombre d'articles, travaux et recherches traitent de réseaux neuronaux artificiels. Ces publications en décrivent les propriétés émanentes, comme la généralisation, l'abstraction, la plasticité, etc. Un nombre considérable de ces travaux sont consacrés à la vision et la reconnaissance visuelle.

Nous nous intéressons à un chemin moins souvent utilisé : le réseau d'ordre supérieur. Ce type de réseau n'est pas nouveau, ni l'implémentation d'une portion de connaissance explicite dans un réseau neuronal, mais peu de travaux sont consacrés à la combinaison des deux.

Le chapitre suivant aborde les fondements théoriques sur les réseaux neuronaux que nous utilisons pour la reconnaissance de formulaires.

INTRODUCTION AUX RESEAUX NEURONAUX

II.1. Introduction.

Quoi de plus naturel que d'essayer de copier le réseau neuronal cérébral si nous voulons copier, ou essayer de comprendre le fonctionnement du cerveau.

Le neurone formel s'inspire du neurone biologique. Il n'en reprend que les fonctionnalités les plus élémentaires. Les aspects biologiques du réseau neuronal ne sont pas d'importance capitale dans ce travail, mais tout lecteur intéressé peut se tourner vers une quantité de livres et de travaux qui les traitent en détail. On pourra lire, par exemple, le chapitre 20 de [Rumelhart-2, 1986] ou aussi [Honet, 1990].

Historiquement, tout commence avec un article de McCulloch et Pitts [McCulloch, 1943] en 1943 sur un modèle de neurone formel. Les années qui suivent voient croître lentement l'intérêt dans le domaine. Dès le début des années '50, la recherche en réseaux neuronaux est en pleine expansion. Un modèle appelé Perceptron, inventé par Rosenblatt en 1957, connaît un succès remarquable tant théorique que pratique. En 1969, Minsky et Papert publient un livre sur les Perceptrons ([Minsky, 1969]), qui refroidit les esprits. Ils démontrent que le Perceptron à une couche n'est apte qu'à résoudre des problèmes très limités. Leur article, ainsi que l'inexistence d'un algorithme d'apprentissage pour des réseaux neuronaux multicouches, arrête la grande majorité des recherches dans le domaine au profit des systèmes experts. Seuls quelques chercheurs ou groupes de chercheurs isolés continuent. En 1982, Hopfield présente le "Crossbar Associative Network" [Hopfield, 1982], mieux connu sous le nom de réseau Hopfield, et relance l'intérêt dans la modélisation neuronale. Un dernier point de repère important est l'apparition du "Parallel Distributed Processing" [Rumelhart, 1986] en 1986, oeuvre de référence pour tout chercheur dans le domaine. Actuellement, nous assistons à une explosion d'activités en matière de réseaux neuronaux. Tant la recherche théorique que les applications sont l'objet d'une attention d'un

public pluri-disciplinaire. Et tant que la limite du réseau neuronal n'est pas en vue, la recherche dans "l'inconnu" continuera à fasciner.

Dans l'histoire, un grand nombre de types de réseaux neuronaux ont vu le jour. Ce n'est nullement le but de ce chapitre de les passer tous en revue. Nous nous limiterons aux types utilisés dans le cadre du mémoire. Une taxonomie des réseaux neuronaux peut être trouvée, entre autres, dans [Nivelles, 1990] , [Wasserman, 1989] et [Rumelhart, 1986].

II.2. Le neurone formel.

Malgré le grand nombre de types de réseaux neuronaux existants, le neurone formel n'a guère évolué. Il est presque toujours formé des éléments de la figure 2-1.

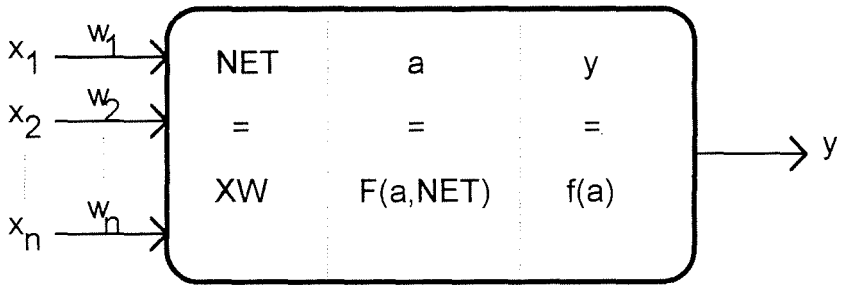

Figure 2-1 : Le neurone formel ou artificiel.

Dans une première partie logique, l'**entrée**, il y a :

X : le vecteur des entrées du neurone. Celles-ci sont soit les valeurs sortantes de neurones d'une autre couche, soit les valeurs d'entrée du réseau.

W : le vecteur des poids de connexion entre les neurones.

NET : la valeur d'entrée totale, calculée à partir de X et W.

La partie logique suivante s'occupe du **calcul du niveau d'activité**. Nous y retrouvons :

a : le niveau ou la valeur d'activation du neurone.

F : la fonction d'activation. Normalement elle est appliquée à la valeur NET uniquement (a = F(NET)), mais parfois l'ancienne valeur d'activité peut intervenir (a = F(a,NET)).

En dernier lieu, la partie logique de **sortie** :

f : la fonction de sortie. Elle est rarement utilisée autrement que comme fonction d'identité (f(x)=x). Elle sert dans les réseaux où un seul neurone par couche peut être actif ou comme fonction de seuil.

y : la valeur de sortie pour le neurone. Elle est calculée comme suit : y = f(a).

Nous ne sommes pas entré ici dans le détail du calcul de NET, de la fonction d'activation et de la fonction de sortie. Il en existe une multitude. On les retrouve par exemple dans [Wasserman, 1989] et [Rumelhart, 1986]. Dans les parties II.4, II.5 et II.6 le détail est fourni dans le cadre du type de réseau traité.

II.3. Classification des réseaux neuronaux.

Pour pouvoir situer un réseau, il est intéressant de donner quelques notions clés.

Une première caractéristique est la **nature des neurones**. Sont-ils binaires ou à valeurs réelles?

Ensuite, le réseau contient-il de la **non-linéarité**? Minsky et Papert ont prouvé que le Perceptron à une seule couche ne sait pas résoudre le ou-exclusif à cause de la linéarité ([Minsky, 1969]). De plus, un réseau multi-couches (voir le paragraphe suivant), du type Perceptron, à fonction d'activation linéaire peut toujours être réduit à un réseau à une seule couche.

En troisième lieu l'**architecture du réseau** intervient. Il peut s'agir d'un réseau à une seule couche de neurones ou bien d'un réseau multi-couches. La couche d'entrée d'un réseau n'est jamais comptée car ses neurones n'exécutent pas de calculs. D'autre part, nous distinguons les couches cachées et la couche de sortie, qui fournit le résultat.

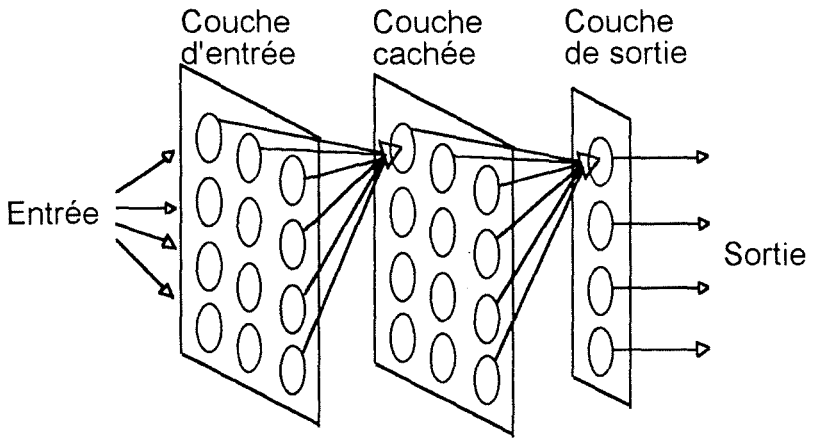

Figure 2-2 : un réseau multi-couches.

Il faut parler ensuite de l'**interconnexion des neurones**. Les neurones peuvent être connectés uniquement aux neurones de la couche suivante. Dans ce cas nous parlons de *réseau feed-forward.* Si les neurones d'une couche sont connectés aux neurones d'une couche inférieure le réseau est appelé *récurrent.* Il se peut que tous les neurones soient connectés à tous les autres neurones du réseaux. Dans ce dernier cas, on l'appelle *réseau entièrement connecté.*

Une autre caractéristique est le **mode de mise-à-jour du niveau d'activation** des neurones. Si la fonction d'activation est calculée pour tous les neurones à chaque unité de temps, il s'agit d'un mode *synchrone.* Si à chaque unité de temps un neurone a une certaine probabilité d'être mise-à-jour, le mode est appelé *asynchrone.* On trouve aussi le mode *mixte,* qui est la combinaison des deux.

La dernière caractéristique que nous abordons est le **type d'apprentissage**. Le plus fréquent est l'*apprentissage supervisé.* Celui-ci consiste à donner au réseau un ensemble de patterns en entrée avec les patterns de sortie attendus. Dans le cas d'*apprentissage non supervisé,* le réseau reçoit uniquement les patterns en entrée. Le réseau apprend les patterns par classification et donne en sortie un pattern imprévisible mais consistant avec le pattern d'entrée fourni. L'utilisateur doit alors faire le lien entre ce pattern de sortie et le résultat attendu. En dernier lieu, on trouve l'*apprentissage auto-supervisé* ou par expérimentation. Le système duquel le réseau fait partie est capable de déterminer par observation ou calcul le résultat de l'activité du réseau et de le corriger automatiquement.

II.4. Le Perceptron.

II.4.1. INTRODUCTION.

McCulloch et Pitts sont les premiers à avoir publié une étude systématique des réseaux neuronaux artificiels ([McCulloch, 1943]). Ils étudiaient des réseaux avec des neurones simples, voir figure 3-1, dont la fonction d'entrée est la somme des entrées pondérées et la fonction d'activation est une fonction binaire à seuil : si la fonction d'entrée dépasse un seuil, alors la sortie vaut 1 sinon elle vaut 0. Ce type de réseau, ainsi que ses nombreuses variations, est connu sous le nom de Perceptron. Pour ce type de réseau un nombre de théorèmes importants ont été prouvés. Voir, par exemple, [Rosenblatt, 1962] ou [Hecht, 1987]. Ces réseaux font du mapping, c'est à dire font une approximation d'une fonction mathématique appliquée aux neurones d'entrée.

II.4.2. CARACTÉRISTIQUES.

Typiquement, un perceptron est un réseau feed-forward, à valeurs binaires ou continues. Il peut être constitué d'une ou de plusieurs couches. Le théorème de Kolmogorov, appliqué aux réseaux neuronaux par Hecht-Nielsen ([Hecht, 1987]), dit qu'un réseau à deux couches et N(2N+1) noeuds peut calculer toute fonction continue à N variables. La mise-à-jour des neurones est en mode synchrone.

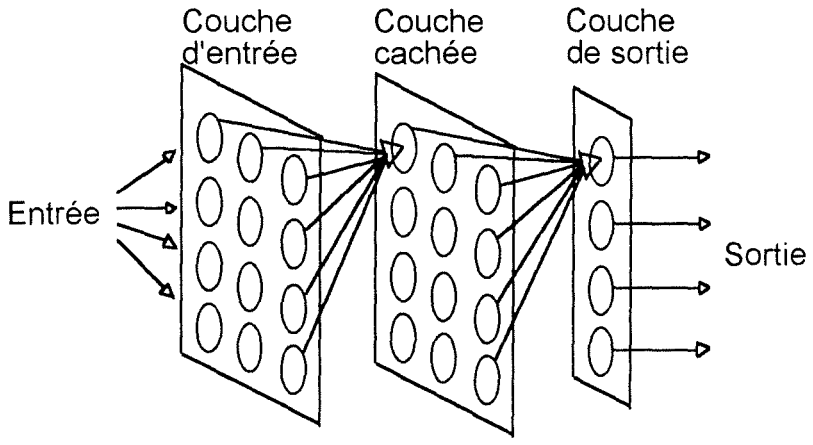

Figure 2-3 : Perceptron à deux couches.

La fonction d'entrée d'un neurone de la couche i est définie par :

La dérivée première de la fonction d'activation est :

$$\frac{\delta F(NET_i)}{\delta NET_i} = y_i(1 - y_i)$$

où y_i est la sortie[3] du neurone.

Pour la *couche de sortie* j, la formule de mise-à-jour est

$$\delta_j = y_j (1 - y_j) (y_{att} - y_j)$$

où y_j est la sortie du neurone de sortie et

y_{att} est la sortie attendue.

Les nouveaux poids de connexion entre la couche de sortie et celle qui lui est juste inférieure sont calculés par

$$w_{j,\ nouveau} = w_{j,ancien} + \eta\ \delta_j\ y_i$$

où $w_{j,\ nouveau}$ est le poids de connexion de la couche de sortie après calcul,

$w_{j,ancien}$ est l'ancien poids de connexion,

η est le taux d'apprentissage,

δ_j est la valeur de correction du poids calculée dans formule précédente et

y_i est le résultat du neurone de la couche inférieure.

Pour la *couche intérieure* nous ne connaissons pas le résultat attendu en sortie. L'écart constaté pour la couche de sortie est propagé par les modifications des poids des noeuds de la couche supérieure par la formule suivante :

$$\delta_i = y_i (1 - y_i) (\Sigma\ \delta_j\ w_j)$$

Ce δ_i est alors inséré dans la formule de modification des poids de connexion de la couche de sortie.

3 Rappelons que la fonction de sortie est l'identité et que donc $y_i=F(NET_i)$.

II.5. Le réseau Hopfield.

II.5.1. INTRODUCTION.

Supposons un problème incluant des contraintes. Classiquement nous entrons ici dans le domaine de la recherche opérationnelle. Les réseaux neuronaux de type Hopfield se prêtent à la résolution de ce type de problèmes. Ils implémentent cependant la résolution sous contraintes "faibles", que l'on souhaiterait satisfaites. Les contraintes "fortes" doivent obligatoirement être satisfaites dans la solution.

Nommons "énergie" la satisfaction aux contraintes par le réseau. Cet énergie est calculée à partir de l'état[4] du réseau à un moment donné. Satisfaire les contraintes de la meilleure façon possible veut donc dire maximiser l'énergie du réseau.

Les neurones symbolisent des hypothèses; les interconnexions expriment la relation entre ces hypothèses. Par exemple : (h1 $\Rightarrow$ h2) est implémenté par un poids positif entre les neurones de h1 et h2.

Nous pouvons donner une valeur par défaut à l'hypothèse par l'introduction d'une valeur appelé "bias".

Il ne reste plus qu'à introduire les observations de l'extérieur. Une valeur, "in_d", indique au neurone *d* si l'hypothèse qui lui correspond est satisfaite dans le problème que l'on traite.

L'énergie est calculée en sommant la participation de chaque neurone :

$$\textbf{Energie} = \sum_{d} (\textbf{net}_d \, \textbf{a}_d)$$

Où $\mathbf{a_d}$: la valeur d'activation du neurone d, et

$\mathbf{NET_d}$: est calculé par la formule suivante :

4 L'état du réseau est l'ensemble des valeurs de tous les neurones du réseau.

$$NET_d = \sum_i w_{id}\, a_i + bias_d + in_d$$

Où w_{id} : le poids d'interconnexion entre le neurone i et le noeud de destination d,

a_i : la valeur d'activation du neurone i,

$bias_d$: la valeur par défaut du neurone d, et

in_d : l'information de l'extérieur.

Les neurones vont maintenant essayer de trouver une valeur pour maximiser la fonction d'énergie. Comme le réseau est récurrent, le processus est dynamique : le réseau recherche un état stable. L'état stable est atteint quand l'état du réseau ne change plus. Ce processus est appelé la relaxation. Il se peut que le réseau ne trouve pas d'état stable pour certaines valeurs de poids de connexion, par exemple par oscillation. Heureusement, Cohen et Grossberg ont prouvé un théorème garantissant la stabilité sous certaines hypothèses, [Cohen, 1983] : premièrement, la matrice doit être symétrique et deuxièmement, les neurones ne peuvent pas être connectés à eux-mêmes.

II.5.2. CARACTÉRISTIQUES.

Un réseau de Hopfield est récurrent, à valeurs binaires ou continues et en mode asynchrone.

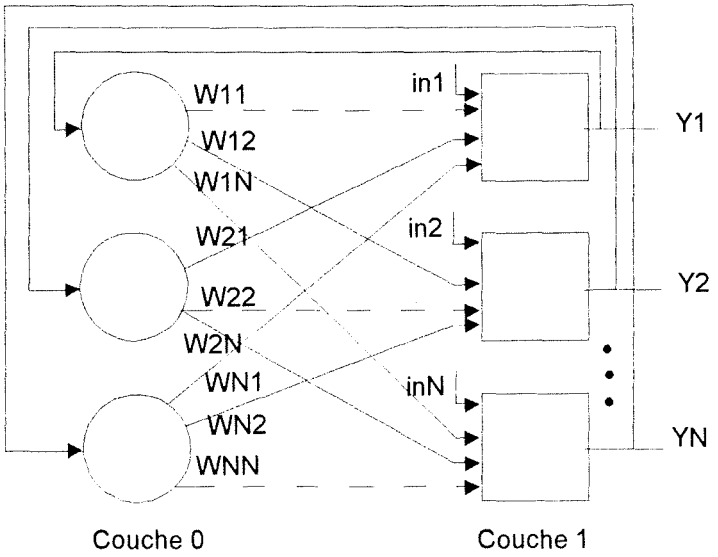

Figure 2-5 : réseau de Hopfield.

Le problème des minima[5] locaux et le recuit simulé.

Nous pouvons nous imaginer la relaxation comme une bille qui, par la gravité, descend jusqu'à trouver un trou duquel elle ne sait plus sortir. La relaxation recherche le minimum le plus proche, or ce qu'il faut est le minimum global. Pour résoudre ce problème une méthode analogue au recuit simulé est appliquée. La méthode a été introduite en 1984 par Kirckpatrick, Gelatt et Vecchi [Kirckpatrick, 1984] basé sur le Metropolis sampling.

L'idée peut être décrite comme suite. Changer en asynchrone quelques poids de connexion par une certaine valeur. La probabilité d'accepter la modification suit la distribution de Boltzmann :

$$P(c) = \frac{1}{1 + e^{-c/kT}}$$

où **P(c)** est la probabilité d'une modification de c dans le niveau d'énergie;

k est une constante analogue à celle de Boltzmann;

T est la "température".

5 L'optimisation par minimalisation est plus adéquate pour l'utilistation de l'image du recuit simulé.

Graduellement la température est réduite et l'énergie se stabilise dans un minimum. De par la méthode, il y a une forte chance de trouver le minimum global.

II.6. Le réseau d'Ordre Supérieur.

II.6.1. INTRODUCTION.

Les réseaux d'0rdre Supérieure ne sont pas récents. Pitts et McCulloch faisaient déjà des recherches en pattern recognition indépendants de l'échelle, la rotation et la translation [Pitts, 1947]. Le problème à ce moment là était surtout l'explosion exponentielle du nombre de calculs à cause des combinaisons. De nos jours, les machines nous permettent d'implémenter de pareils algorithmes avec des temps de réponse admissibles [Reid, 1990].

II.6.2. CARACTÉRISTIQUES.

Le neurone formel est pratiquement égal à celui du Perceptron. Il s'agit d'un réseau feed-forward, continu à une seule couche et mise-à-jour en synchrone.

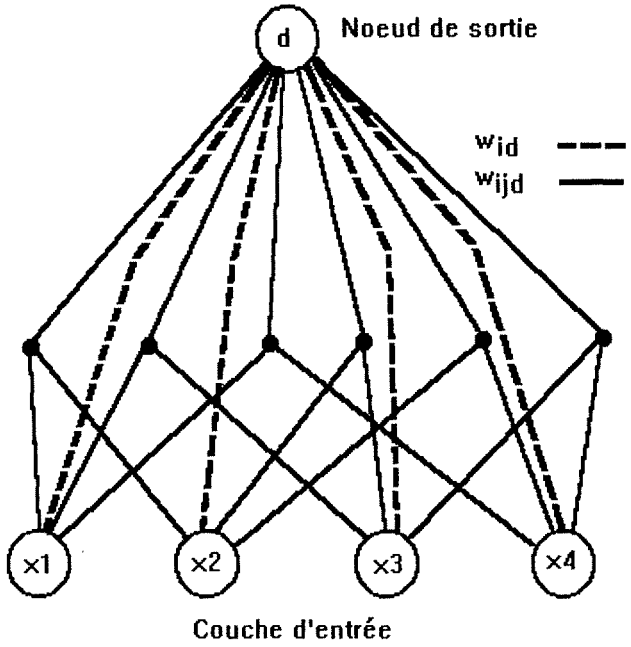

Figure 2-6 : un réseau de second ordre.

De façon générale, la fonction d'entrée totale est de la forme :

$$NET_d = \sum_i w_{id} x_i + \sum_i \sum_j w_{ijd} x_i x_j + \sum_i \sum_j \sum_k w_{ijkd} x_i x_j x_k + \ldots$$

La fonction d'activation est une fonction non-linéaire; par exemple, la fonction logistique.

II.6.3. L'INVARIANCE.

Il est inutile de faire découvrir au réseau des propriétés de patterns en entrée par de longues sessions d'apprentissage si nous pouvons insérer cette connaissance immédiatement. Pour implémenter une invariance d'échelle, de rotation et de translation dans un réseau, il faut trouver une caractéristique indubitable du pattern. Prenons des droites comme exemple. Quelle est une caractéristique d'une droite, indépendamment de sa translation et de son échelle? - L'angle de la droite par rapport à une référence générale.

Supposons une droite définie par les points *p1* et *p2* ; avec p1 = (x1,y1) et p2 = (x2,y2). Le coefficient angulaire de la droite est calculée par **(y2-y1)/(x2-x1)**. Toutes les droites parallèles ont le même coefficient angulaire et donc là l'invariance de translation est satisfaite. L'invariance d'échelle est satisfaite:
Supposons une paire de points d'une droite: p1 = (x1,y1) et p2 = (x2,y2).
Multipliée par l'échelle "e": p1' = (e x1, e y1) et p2' = (e x2, e y2).
Le coefficient angulaire : (e y2 - e y1) / (e x2 - e x1)
= e (y2 - y1) / e (x2 - x1)
= (y2 - y1) / (x2 - x1)
Ce dernier est égal au coefficient angulaire de la droite d'origine.

Comment traduire ceci dans un réseau? Il suffit de donner à toutes les paires de points du bitmap d'entrée qui ont le même coefficient angulaire, le même poids.

$$\frac{(y_j - y_i)}{(x_j - x_i)} = \frac{(y_{j'} - y_{i'})}{(x_{j'} - x_{i'})} \Rightarrow w_{ijd} = w_{i'j'd}$$

... pour tout neurone **d** de la couche de sortie.

Les neurones **i,j, i'** et **j'** sont des neurones du bitmap d'entrée. Le noeud **i** est défini par ses coordonnées (x_i,y_i).

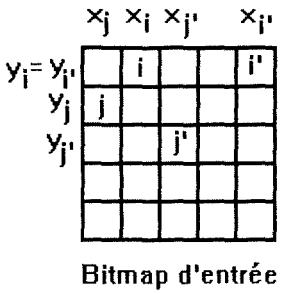

Figure 2-7 : invariance de translation et échelle.

La méthode nécessite un réseau d'ordre deux.

L'invariance aux trois types de transformations est obtenue par l'utilisation d'un triangle inscrit dans le pattern. Les trois angles intérieures du triangle ne changent pas si le triangle est transformé. Il faut donc un réseau d'ordre trois dans lequel les poids sont égaux pour les triplets d'entrée qui forment un triangle avec les mêmes angles intérieurs.

$$w_{\alpha\beta\gamma d} = w_{\beta\gamma\alpha d} = w_{\gamma\alpha\beta d}$$

où $w_{\alpha\beta\gamma d}$ est le poids de la connexion entre le triplet (i, j, k) dont les angles intérieures sont dans l'ordre relatif[6] : $\alpha\beta\gamma$.

L'ordre des angles est important. Si nous interchangeons l'ordre relatif des angles nous obtenons également l'invariance à la symétrie de ligne : $w_{\alpha\beta\gamma d} = w_{\beta\gamma\alpha d} = w_{\gamma\alpha\beta d} = w_{\alpha\gamma\beta d} = w_{\gamma\beta\alpha d} = w_{\beta\alpha\gamma d}$.

II.6.4. L'APPRENTISSAGE.

L'apprentissage du réseau neuronal d'ordre supérieur diverge sensiblement de celui du perceptron à deux couches. Le réseau neuronal d'ordre supérieur ne contient qu'une seule couche et on peut

6 l'ordre dans lequel les angles sont mesurés (dans le sens de la montre ou non) et quel angle est mesuré d'abord n'a aucune importance.

donc y appliquer une règle d'apprentissage simple. Le perceptron à couches cachées nécessite le "back-propagation" (voir II.4.3).

L'apprentissage du réseau neuronal d'ordre supérieur est supervisé et se fait par la règle delta :

$$\Delta w_{ijd} = (t_d - y_d)\, x_i\, x_j$$

où w_{ijd} est le poids de la connexion de la paire de neurones i et j vers le noeud d.

t_d est la sortie attendue;
y_d est la sortie obtenue;
x_i et x_j sont les entrées.

II.6.5. L'EXPLOSION COMBINATOIRE.

Nous pouvons considérer les deux cas : la formule générale d'ordre k pour n neurones d'entrée ou la forme simplifiée pour l'ordre k. La forme simplifiée ne tient compte que du terme contenant k sommations imbriquées : combinaison de k éléments pris parmi n.

La formule générale : $\sum_k C_n^k = \sum_k \frac{n!}{(n-k)!\, k!}$

La formule simple : $C_n^k = \frac{n!}{(n-k)!\, k!}$

ORDRE	GENERALE	SIMPLE
1	n	n
2	$(n^2 + n) / 2$	$(n^2 - n) / 2$
3	$(n^3 + 5n) / 6$	$(n^3 - 3n^2 + 2n) / 6$
4	$(n^4 - 2n^3 + 11n^2 + 14n) / 24$	$(n^4 - 6n^3 + 11n^2 - 6n) / 24$

Nous pouvons considérer que la complexité du calcul et l'espace mémoire nécessaire est intuitivement de l'ordre de $o * n^k$,

o est le nombre de neurones dans la couche de sortie,

n est le nombre de neurones dans la couche d'entrée, et

k est l'ordre du réseau.

FEATURE MATCHING

III.1. Introduction.

Dans ce chapitre nous explorons les possibilités d'implémenter la reconnaissance de formulaires, soit immédiatement sur le bitmap[7], soit en utilisant les caractéristiques[8] obtenues par les procédures décrites dans le chapitre I.

La reconnaissance du formulaire se base surtout sur l'association de patterns. Lors de l'apprentissage, la connaissance des formulaires est distribuée sur un grand nombre de poids (les connexions entre les neurones) . Quand nous présentons un bitmap au réseau, les poids calculent un pattern de sortie qui a été appris auparavant. Grâce au nombre de connexions, la présence de bruit n'influence que partiellement et graduellement la reconnaissance. Ceci garantit au produit fini une robustesse demandée . Comment apprendre à reconnaître des formulaires qui ont été soumis à une transformation? Une solution possible est de faire apprendre un nombre très élevé de cas et laisser le réseau neuronal construire ses propres règles d'invariance. Cette méthode est peu pratique et peu fiable. Mieux vaut instruire le réseau en y incorporant une loi générale. Nous y reviendrons dans le chapitre suivant.

Nous utilisons deux types de réseaux très connus dans le domaine: un Perceptron et un réseau Hopfield. Nous montrons que dans leurs formes élémentaires, ils ne sont pas adéquats pour la reconnaissance en présence de transformations.

III.2. Perceptron.

III.2.1. DESCRIPTION DU RÉSEAU.

Nous choisissons un réseau à deux couches, une couche cachée et une couche de sortie, la couche d'entrée ne comptant pas parce qu'elle ne fait pas de calculs.

7 Cf. page 7
8 D'où le titre du chapitre : "feature matching" (correspondance de caractéristiques.)

Pour des raisons d'espace mémoire, nous devons limiter la taille du bitmap donné, c'est à dire de la couche d'entrée. La division est la suivante : la couche d'entrée compte 81 neurones (bitmap de 9 * 9 pixels); la couche cachée compte 18 neurones et la couche de sortie 9. Le choix du nombre de neurones cachés est toujours un peu arbitraire. Plus il est élevé, plus le réseau peut retenir de patterns différents. Mais le nombre de neurones cachés augmente également le nombre de calculs. Il faut donc faire des compromis selon les performances et l'espace mémoire des machines employées.

III.2.2. ALGORITHME DU PERCEPTRON.

Deux cas peuvent se présenter :

a.) Nous appliquons un pattern au réseau pour avoir un résultat.

Pour chaque couche calculée : (couche cachée, puis couche de sortie)

Pour chaque neurone de la couche :

Calculer le niveau d'activation du neurone selon la formule vue au point II.4.2. :

Les neurones de sortie contiennent alors un pattern de sortie appris. Il correspond au pattern que le réseau a cru reconnaître en entrée.

b.) Le réseau doit apprendre un ensemble de patterns.

NCYCLES fois :

Erreur_sur_cycle = 0,

Pour chaque pattern :

Calculer la sortie comme au point a.

Calculer l'erreur pour ce pattern par la formule :

$$\text{Erreur\_pattern} = \Sigma_j (t_j - y_j)^2$$

où t_j est le $j^{ième}$ élément du pattern attendu et

y_j est le neurone de sortie j.

Erreur_sur_cycle = Erreur_sur_cycle + Erreur_pattern.

Corriger le poids des connexions vers la couche de sortie.

Corriger le poids des connexions vers la couche cachée.
(Voir II.4.3.)

Si Erreur_sur_cycle < ERREUR_ACCEPTABLE, alors s'arrêter.

NCYCLES et ERREUR_ACCEPTABLE sont des paramètres du système dépendants des patterns à apprendre.

Pourquoi faire apprendre les patterns en cycles et ne pas faire apprendre les patterns individuellement? Apprendre un nouveau pattern induit des changements de poids considérables au début. Ceux-ci pourraient modifier la reconnaissance d'un pattern déjà appris. En apprenant en cycles, tous en même temps, les modifications diminuent pour tous les patterns.

L'algorithme du Perceptron est bien connu et les implémentations sont nombreuses. Une implémentation flexible existante est abordée dans le point suivant.

III.2.3. UNE IMPLÉMENTATION DU PERCEPTRON.

J'ai choisi d'utiliser les programmes de McClelland et Rumelhart [Rumelhart-3, 1988]. Ils font partie d'un ensemble d'implémentations de nombreux réseaux neuronaux. Leur but est purement pédagogique et met donc l'accent sur la clarté plus que sur la performance.

III.2.3.1. Description du Perceptron.

```
définitions:
nunits 108
ninputs 81
noutputs 9
nepochs 1000
ecrit 0.01
end
network:
%r 81 18 0 81
%r 99 9 81 18
end
biases:
%. 81 27
end
```

Figure 3-1 : Description du Perceptron utilisé.

La partie *définitions* contient les constantes du réseau.

Nunits est le nombre total de neurones dans le réseau.

Ninputs et *noutputs* indiquent respectivement le nombre de neurones dans la couche d'entrée et dans la couche de sortie.

Les constantes NCYCLES et ERREUR_ACCEPTABLE de l'algorithme ci-dessus, sont définies respectivement par *nepochs* et *ecrit*.

Dans la partie *network* les connexions entre couches sont explicitées.

Pour lire plus facilement les lignes qui débutent par le signe "%", il est utile de savoir que les neurones du réseau sont numérotés de 0 à 107. Les noeuds de 0 à 80 font partie de la couche d'entrée. La couche cachée va du numéro 81 au 98, 18 neurones en tout, et finalement la couche de sortie : les 9 unités de 99 à 107.

La première ligne décrit les connexions entre la couche d'entrée et la couche cachée. On doit lire cette ligne comme suite : "Il existe des connexions, initialisées à des valeurs aléatoires (r) entre -1 et +1, entre les 18 neurones, comptant à partir du noeud numéro 81 (le premier 81), et les 81 unités à partir du noeud 0."

La deuxième ligne décrit, de la même manière, les connexions entre la couche cachée (81 18) et la couche de sortie (99 9). Les poids des connexions sont également initialisés à une valeur aléatoire.

Initialiser les poids à 0 ne permettra pas au réseau d'apprendre puisque le d est calculée par une multiplication ayant l'ancien poids comme opérande.

La dernière partie, *biases* , initialise les "tendances" des couches calculées. Le "." signifie que les valeurs sont mises à 0 et non modifiables par apprentissage. Les "tendances" sont expliqués dans le point III.2.3.3.

III.2.3.2. Paramètres propres aux patterns à apprendre.

```
get network perceptr.net
get patterns learn1.pat
set lflag 1
set param lrate 0.9
set param momentum 0.0
strain
```

Figure 3-2 : Fichier d'initialisation du réseau.

Ce fichier décrit les étapes à parcourir pour initialiser un réseau avant l'apprentissage et la reconnaissance des patterns.

Les deux premières lignes font des lectures de fichiers. D'abord le réseau est initialisé (voir III.2.3.1.), puis les patterns à apprendre ou à reconnaître sont lus.

Nous supposons être dans un mode d'apprentissage. Pour indiquer ceci nous mettons un drapeau d'apprentissage *lflag* à 1.

Dans les formules de correction des poids intervient un taux d'apprentissage h. Le paramètre *lrate* correspond exactement à cela. L'apprentissage se fait en minimisant l'erreur. Le taux d'apprentissage influence la taille de la modification. Si la modification est trop grande, elle peut faire dépasser le minimum sans pour autant passer en dessous d'ERREUR_ACCEPTABLE (voir figure 3-3) et ainsi mettre le réseau dans une situation d'oscillation.

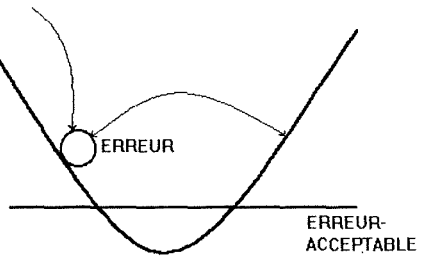

Figure 3-3 : Oscillation dans l'apprentissage.

Il faut un compromis car si le taux est trop petit, l'apprentissage peut s'éterniser. Par conséquent, il est intéressant de prendre h le plus grand possible, mais en même temps en dessous de 1. Empiriquement, une valeur de 0.9 a donné d'excellents résultats.

Le paramètre *momentum* est lié à une méthode d'accélération d'apprentissage. Pour rester dans le cadre du Perceptron classique ce paramètre doit être initialisé à 0. Nous y reviendrons dans le point suivant.

Finalement, la commande *strain* ordonne l'apprentissage des patterns lus. L'apprentissage peut être séquentiel ou permuté. Dans le cas séquentiel, les patterns du cycle sont parcourus dans l'ordre lu. Dans le cas permuté, l'ordre d'apprentissage des patterns est aléatoire. Ceci peut parfois aider à briser des symétries et par conséquent des oscillations de poids de connexion à l'intérieur d'un cycle. Ces cas sont rares mais il m'est arrivé que le réseau ne voulait pas apprendre un

pattern à cause de ce type d'oscillation. Changer le mode d'apprentissage suffisait. Une autre solution est de réinitialiser les poids aléatoires des connexions. Dans ce cas il faut évidemment réapprendre tous les patterns.

III.2.3.3. Méthode du momentum.

Le back-propagation, comme nous l'avons proposé jusqu'à présent, prend souvent beaucoup de cycles avant d'apprendre une série de patterns, même petits. Une méthode pour accélérer ce processus sans provoquer d'oscillation est d'inclure dans les formules du back-propagation un terme α dit "momentum" [Rumelhart-3]. La nouvelle formule devient alors :

$$\Delta w_{ij,nouveau} = \alpha \, \Delta w_{ij,ancien} + \eta \, \delta_j \, y_i$$

où α est le terme de "momentum",
Δw_{ij} est la modification appliquée au poids de la connexion du neurone i vers le noeud j,
η est le taux d'apprentissage,
δ_j est le terme de correction d'erreur, et
y_i est la sortie du neurone i.

Ajouter le terme de "momentum" permet de mieux guider l'erreur dans sa descente vers le minimum. Le facteur "momentum" ajoute une inertie aux modifications des poids, eliminant ainsi fortement les variations brusques, génératrices d'oscillation.

La méthode du momentum réduit, en moyenne, le nombre de cycles nécessaires à l'apprentissage à 25% de celui du back-propagation simple. Dans des cas très favorables que j'ai rencontrés, elle le réduisait à 5%.

La "tendance" (bias) par contre n'a dans aucun cas influencé de façon convaincante le nombre de cycles nécessaires à l'apprentissage. Il peut servir néanmoins à éviter des biais mathématiques. La multiplication a 0 comme élément absorbant. Ainsi, le réseau peut être paralysé. Pour la même raison, les poids sont initialisés à une valeur aléatoire non nulle. La "tendance" peut être vue comme une connexion d'un neurone qui est toujours actif. Et cette connexion apprend comme toutes les autres connexions.

III.2.4. RECONNAISSANCE SUR BITMAP.

III.2.4.1. Apprentissage des patterns.

Dans cette partie nous examinons la reconnaissance de formulaires, nous basant sur les lignes et les intersections, en gardant les bitmaps. Nous avons pris les quatre bitmaps suivants comme exemples. Chacun représente une intersection de type "T" dans une des quatre directions du vent.

```
        T1                  T2
0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0   0 0 0 1 1 0 0 0 0
0 1 1 1 1 1 1 0 0   0 0 0 1 1 0 0 0 0
0 1 1 1 1 1 1 0 0   0 0 0 1 1 0 0 0 0
0 1 1 0 0 0 0 0 0   0 0 0 1 1 0 0 0 0
0 1 1 0 0 0 0 0 0   0 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0   0 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 1 1 1 1

        T3                  T4
0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0   0 0 1 1 1 1 1 1 0
0 0 0 0 0 0 1 1 0   0 0 1 1 1 1 1 1 0
0 0 0 0 0 0 1 1 0   0 0 0 0 1 1 0 0 0
0 0 1 1 1 1 1 1 0   0 0 0 0 1 1 0 0 0
0 0 1 1 1 1 1 1 0   0 0 0 0 1 1 0 0 0
0 0 0 0 0 0 1 1 0   0 0 0 0 1 1 0 0 0
0 0 0 0 0 0 1 1 0   0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0

1 1 1 1 0 0 0 0 0   1 1 1 1 1 1 1 1 1
```

Figure 3-4 : Les patterns de référence.

Chaque pattern est défini en trois parties : le nom, le pattern et le résultat attendu.

Le nombre limité de patterns ainsi que leur taille réduite permet un apprentissage relativement rapide. Il permet aussi d'apprécier l'influence du taux d'apprentissage η.

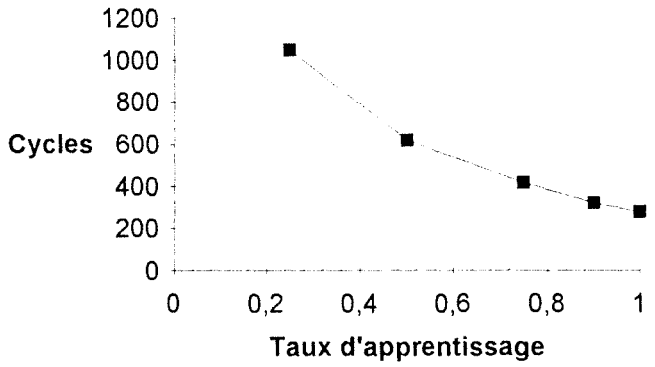

Figure 3-5 : Influence du taux d'apprentissage η

III.2.4.2. La translation.

Nous nous intéressons à la reconnaissance d'un pattern en présence d'une translation. La translation apportée est un simple décalage du "T" vers la droite et le bas. Nous le déplaçons de deux rangées et de deux colonnes.

TRANSLATION

0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0
0 0 0 1 1 0 0 0 0
0 0 0 1 1 1 1 1 1
0 0 0 1 1 1 1 1 1
0 0 0 1 1 0 0 0 0
0 0 0 1 1 0 0 0 0

Figure 3-6 : Translation du "T".

Le résultat obtenu après application de ce pattern au réseau qui a appris les quatre patterns de référence de la figure 3-4.

Pattern name : TRANSLATION

Output pattern :	56	51	58	58	14	95	93	89	92
Train. pattern :	0	0	0	0	0	0	0	0	0

Error : 4.7299

Figure 3-7 : Résultat de la reconnaissance.

Le résultat de l'application du pattern au réseau est la liste de neuf valeurs derrière le "output pat." Ces nombres représentent les valeurs

d'activation des neuf neurones de sortie. Nous travaillons avec un réseau à valeurs continues entre -1 et +1. Les nombres sont des centièmes. La valeur du dernier neurone est donc 0.92. Le "training pat." est le vecteur de résultat attendu. Evidemment, le réseau n'en tient pas compte lors du calcul du résultat. Le pattern attendu permet d'apprécier l'erreur commise.

L'erreur, calculée par la somme des carrés des différences, se trouve dans un intervalle de 0.0 à 9.0

Clairement le Perceptron est incapable de percevoir la translation du pattern T1. Les quatres dernières valeurs indiquent que le réseau donne une nette préférence aux pattern T2 et T4. Les quatres valeurs de début montrent une indécision quant au pattern exact. La cinquième valeur, celle du milieu, est intéressante. Le neurone du milieu n'est activé que dans le cas du pattern de référence T4. Le 0.14 indique une indécision du neurone. On peut l'interpréter aussi comme : "il y a une forte chance (à peu près 85%) que le bon pattern ait une valeur 0 dans le cinquième neurone." Le nombre d'activations sur le nombre total de patterns de référence, est pour le neurone une sorte d'indication de tendance. Cette interprétation de la valeur de sortie est également valable pour les autres neurones. C'est pourquoi nous avons indiqué que les quatres neurones de début sont indécis. Il n'y a aucune explication immédiate pourquoi des neuf neurones, celui du milieu est le seul à avoir une tendance vers le bas (par rapport au 0.25, sa valeur par défaut).

III.2.4.3. Echelle.

Nous soumettons au réseau deux patterns dérivés du pattern de référence T1. Le premier est réduit à 70% de sa taille originale. Le deuxième à une réduction de 50%.

SCALE70	SCALE50
0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0	0 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0	0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 0	0 0 0 0 0 1 0 0 0
0 0 0 1 1 0 0 0 0	0 0 0 0 0 1 1 1 0
0 0 0 1 1 0 0 0 0	0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0

Figure 3-8 : Réduction à 70% et à 50%.

Dans les deux cas la reconnaissance a de nouveau échoué.

```
Pattern name : SCALE70
Output pattern :    34    25    36    29    10    92    91    92    90
Train. pattern :     0     0     0     0     0     0     0     0     0
Error :  3.7684
```

Figure 3-9 : Reconnaissance du SCALE70.

De nouveau nous constatons une affinité pour le pattern de référence T2. Nous pourrions expliquer ceci en remarquant que le SCALE70 et le T2 ont 10 neurones actifs[9] sur les seize en commun. Dans les trois autres cas, l'intersection est de 4, 7, et 5 neurones actifs, respectivement pour le T1, T3 et T4. Le SCALE70 "ressemble" donc le plus au T2.

Cette intuition est apparemment contredite par l'exemple du SCALE50 :

```
Pattern name : SCALE50
Output pattern :    69    68    72    74    13    71    73    70    70
Train. pattern :     0     0     0     0     0     0     0     0     0
Error :  4.1047
```

Figure 3-10 : Reconnaissance du SCALE50.

Ici le choix serait plutôt le T4. Seulement, il ne possède que 2 éléments actifs en commun avec le SCALE50, tandis que le T2 et le T3 ont tous les deux 3 valeurs à 1 en commun.

Comment expliquer ceci? Il y a l'influence de la toute petite différence entre avoir 2 et avoir 3 neurones en commun. Ensuite il faut noter que le réseau n'est pas du tout affirmatif dans son choix. Les valeurs des neurones restent près des valeurs par défaut. Donc il y a compétition, à armes égales, entre les patterns T2, T3 et T4. Le cinquième neurone de sortie donne une indication intéressante. En s'approchant de 0, il défavorise fortement la possibilité T4, c'est à dire la solution 111 111 111. Ceci indique que la compétition réelle est entre T2 et T3. Or la combinaison de 1111 0 0000 et 0000 0 1111 se rapproche effectivement de 0.5 0.5 0.5 0.5 0 0.5 0.5 0.5 0.5. L'explication intuitive est sauvegardée.

9 Nous entendons par neurone "actif", un élément dont la valeur vaut 1 ou l'équivalent.

III.2.4.4. Rotation.

Le dernier paramètre dans la liste des transformations est la rotation. Pour l'être humain il s'agit de la moins facile des trois. Pencher la tête pour reconnaître quelque chose n'est pas rare. De la même façon, si un texte est écrit à l'envers, dessus dessous, nous retournons mentalement les lettres pour pouvoir les lire.

ROTATION
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 1 1 1 0
0 1 0 0 1 1 1 0 0
1 1 1 1 1 1 0 0 0
0 1 1 1 1 0 0 0 0
0 0 1 1 1 0 0 0 0
0 0 0 1 1 1 0 0 0
0 0 0 0 1 0 0 0 0

Figure 3-11 : Pattern d'un T après rotation.

Avec les quatre patterns T1 à T4, nous sommes incapables de situer le pattern d'origine. Ajoutons un pattern : le T10, une ligne diagonale.

T10
0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 1 1 1
0 0 0 0 0 1 1 1 0
0 0 0 0 1 1 1 0 0
0 0 0 1 1 1 0 0 0
0 0 1 1 1 0 0 0 0
0 1 1 1 0 0 0 0 0
1 1 1 0 0 0 0 0 0
1 1 0 0 0 0 0 0 0

1 1 1 0 0 0 1 1 1

Figure 3-12 : Pattern T10, une ligne diagonale.

Et comme prévu intuitivement, le réseau considère que ce pattern ressemble le plus au pattern de l'input. Mais en même temps le résultat (Figure 3-13) indique que la correspondance est loin d'être parfaite.

```
Pattern name : ROTATION
Output pattern :   46   55   46    8    2   12   79   72   73
Train. pattern :    0    0    0    0    0    0    0    0    0
Error :  2.4433
```

Figure 3-13 : Reconnaissance du pattern ROTATION.

III.2.4.5. Conclusion.

Le Perceptron n'a réussi aucun test. Avec des exemples simples, nous avons montrés que le Perceptron ne prête attention qu'à la correspondance individuelle entre les neurones des différents bitmaps. Cette correspondance est purement spatiale.

Ceci nous mène à la conclusion que le Perceptron n'est pas adéquat dans le cadre de l'objectif donné. Toute reconnaissance des formulaires, sur base des lignes et en utilisant les bitmaps, est vouée à l'échec si la moindre transformation a eu lieu. Dans la pratique les correspondances parfaites sont malheureusement inexistantes.

III.2.5. RECONNAISSANCE PAR LES CARACTÉRISTIQUES.

III.2.5.1. Introduction.

Supposons maintenant que l'on obtienne les caractéristiques du formulaire à reconnaître par le même chemin que la méthode classique, c'est à dire non-neurale. Les lignes sont extraites du bitmap d'entrée. Ensuite, de façon procédurale, le frame du formulaire est pré-traité pour en extraire finalement les caractéristiques. Dans la méthode classique, anticipons un peu, ces caractéristiques sont ensuite comparées à celles des formulaires de référence, les modèles, stockés dans une base de données.

Quelle que soit la méthode de recherche dans la base de donnée, le temps de réponse est dépendant de sa taille. Si on est limité aux caractéristiques mentionnées, il faut parcourir toute la base de données pour trouver la meilleure correspondance.

Un réseau neuronal du type Perceptron par contre, calcule en un temps constant le résultat correspondant aux données d'entrée, quel que soit le nombre de patterns que le réseau ait appris. Le nombre de formulaires que le réseau doit pouvoir apprendre intervient évidemment dans la configuration du réseau, et donc dans le temps de calcul. Une

des contraintes imposées au produit fini est qu'il soit rapide. Le délai de réponse est uniquement imposé pour la restitution des connaissances existantes et non du point de vue acquisition des connaissances. Un réseau Perceptron couplé au processus est donc envisageable. Un tel système hybride est-il acceptable pratiquement?

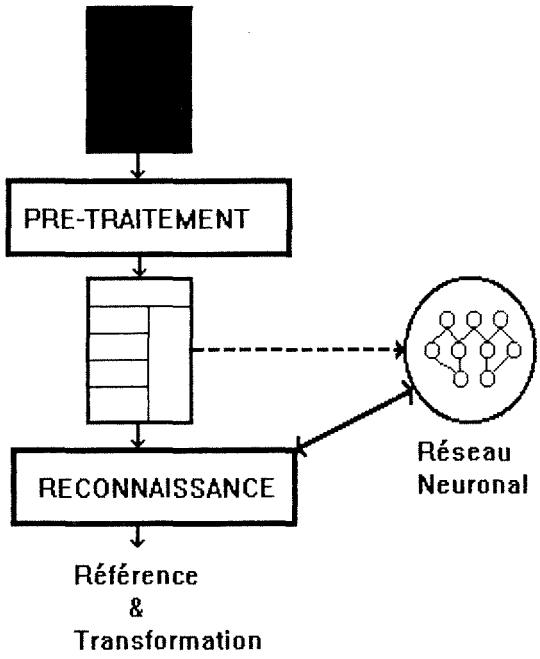

Figure 3-14 : Un processus classique avec une mémoire neuronale.

III.2.5.2. Représentation des caractéristiques.

Les caractéristiques des formulaires que le réseau doit retenir sont des nombres. Rappelons que, dans le pré-traitement, les occurrences de chaque type d'intersection sont comptées.

Nous pouvons introduire ces nombres de différentes manières dans le réseau. Soit nous attribuons les valeurs en base dix aux neurones d'entrée[10], soit nous les convertissons en binaire pour garder des neurones binaires. Examinons les deux cas.

Un réseau acceptant des valeurs **non binaires** peut être réduit considérablement en taille. Un nouveau Perceptron a été créé, contenant neuf neurones dans la couche d'entrée, neuf dans la couche cachée et neuf en sortie.

[10] Seuls les éléments de la couche d'entrée possèdent des valeurs au dessus de +1

Le test s'est effectué avec les valeurs de la base de données créées par la méthode classique (Figure 3-15). Les caractéristiques des cinq formulaires dans les quatre directions du vent constituent les patterns de référence. Dans cet exemple nous rencontrons déjà quelques grandes limites d'un Perceptron.

Premièrement, deux patterns identiques ne peuvent être appris avec des résultats différents. La méthode classique, par contre, peut fournir une liste de résultats équivalents. Le réseau apprend tous les patterns. Dans le cas de patterns identiques, les neurones de sortie qui sont différents se voient attribués des valeurs par défaut. Par exemple, si un des deux patterns identiques a un neurone de sortie à 1 et l'autre à 0, le réseau apprend pour les deux une valeur proche de 0.5 en sortie. Dès lors l'erreur sur les deux patterns est d'au moins[11] 0.5. Si le seuil d'apprentissage, ERREUR_ACCEPTABLE, est inférieure à 0.5 l'apprentissage ne s'arrête évidemment jamais.

A cause de cela nous avons éliminé les patterns identiques de notre exemple. Il reste quinze patterns à apprendre. Ils sont décrits comme suit :

nom du formulaire
valeurs des caractéristiques
pattern de sortie attendu.

```
E111_0                  INTERN_90               INVOICE_270
8 8 11 7 5 1 2 5 0      3 3 3 3 0 10 0 10 0     1 1 1 1 1 1 0 1 6
0 0 0 0 0 0 0 1 1       1 0 0 0 0 0 1 1 0       0 0 1 0 1 1 0 0 0

E111_90                 MCQ_0                   PERSON_0
7 8 8 11 5 5 1 2 0      6 6 6 6 5 5 5 5 5       2 1 1 2 1 5 1 0 0 5
1 0 0 0 0 0 0 1 1       0 0 0 0 0 1 1 0 0       0 0 0 1 1 0 0 0 0

E111_180                INVOICE_0               PERSON_90
11 7 8 8 2 5 5 1 0      1 1 1 1 1 1 1 0 6       2 2 1 1 0 15 1 0 5
0 1 0 0 0 0 0 1 1       0 0 0 0 1 1 0 0 0       1 0 0 1 1 0 0 0 0

E111_270                INVOICE_90              PERSON_180
8 11 7 8 1 2 5 5 0      1 1 1 1 0 1 1 1 6       1 2 2 1 0 0 15 1 5
0 0 1 0 0 0 0 1 1       1 0 0 0 1 1 0 0 0       0 1 0 1 1 0 0 0 0

INTERN_0                INVOICE_180             PERSON_270
3 3 3 3 10 0 10 0 0     1 1 1 1 1 0 1 1 6       1 1 2 2 1 0 0 15 5
0 0 0 0 0 0 1 1 0       0 1 0 0 1 1 0 0 0       0 0 1 1 1 0 0 0 0
```

Figure 3-15 : Les patterns de référence des formulaires.

[11] $2 * (1 - 0.5)^2$

E111REAL
6 6 9 7 7 2 2 5 1

Figure 3-16 : Pattern à reconnaître.

Nous introduisons le pattern du formulaire rempli E111 (Figure 3-16) et le réseau fournit le résultat que voici :

Pattern name :	E111REAL								
Input pattern :	600	600	900	700	700	200	200	500	100
Hidden pattern :	0	01	99	0	99	1	99	99	0
Output pattern :	59	0	1	0	0	11	99	93	0
Training pattern :	0	0	0	0	0	0	0	100	100
Error :	2.3578								

Figure 3-17 : Résultat de reconnaissance.

Clairement le réseau reconnaît le pattern INTERN_0, et donne le INTERN_90 comme deuxième choix très probable. Il a également le pattern MCQ_0 en réserve en donnant 0.11 pour le sixième neurone. Inutile d'insister sur le fait qu'aucune des trois propositions n'est la bonne.

Nous voici au deuxième problème du Perceptron : l'interprétation des résultats. Le réseau tient sa promesse de délai de réponse, mais le résultat n'est pas toujours univoque. Nous venons d'en avoir l'exemple. Si nous appliquons une fonction de sortie, une fonction à seuil, sur les neurones de sortie, le résultat devient unique. Nous perdons alors l'avantage de pouvoir interpréter les résultats. Il est avantageux d'avoir plusieurs candidats "reconnus". De plus, comme nous l'avons déjà vu, les valeurs des noeuds de sortie donnent une sorte d'appréciation des candidats. Il est important de voir que l'interprétation des résultats est entièrement dépendante de la représentation des patterns de sortie. Dans les exemples utilisés, chaque pattern de référence a une représentation simple en sortie, facilitant l'interprétation. Comment traduire l'activation des neurones de sortie en une liste ordonnée de candidats formulaires? L'interprétation étant propre à chaque réseau, il semble intéressant d'y attacher un interpréteur automatique (système expert?)

Faire reconnaître des patterns en base dix s'est avéré impossible. Qu'en est-il des **représentations binaires**[12]?

Si nous acceptons la thèse que deux patterns se "ressemblent" s'il y a une correspondance importante d'éléments actifs, il ne suffit pas de simplement traduire les nombres de base dix vers la base deux et puis de faire apprendre ceux-ci au réseau. Par exemple, un huit binaire (0 0 1 0 0 0) ne "ressemble" en rien au sept binaire (0 0 0 1 1 1).

Par contre, un huit (0 0 1 1 1 1 1 1 1 1) "ressemble" au sept (0 0 0 1 1 1 1 1 1 1). Vérifions la thèse dans ce cas :

E111_0	INTERN_0
0 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 1 1 1
0 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 1 1 1
1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 1 1 1
0 0 1 1 1 1 1 1 1 1	0 0 0 0 0 0 1 1 1
0 0 0 0 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1	1 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 1	0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1	0 0 0 0 0 0 1 1 0

Figure 3-18 : Deux patterns parmi les quinze appris (voir figure 3-15).

E111REAL
0 0 0 1 1 1 1 1 1
0 0 0 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
0 0 1 1 1 1 1 1 1
0 0 1 1 1 1 1 1 1
0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 1 1
0 0 0 0 1 1 1 1 1
0 0 0 0 0 0 0 0 1

Figure 3-19 : Pattern à reconnaître.

```
Pattern name : E111REAL
Output pattern :    4    0    0    0    0    1   11   98   83
Train. pattern :    0    0    0    0    0    0    0  100  100
Error :  2.4433
```

Figure 3-20 : Résultat de la reconnaissance.

[12] Le binaire veut dire : base 2, dans le paragraphe qui suit; et utilisation, dans les patterns d'entrée, de nombres 0 et 1 uniquement, dans le paragraphe suivant.

Il n'y a pas de doute sur la reconnaissance. Le réseau a bien retrouvé le pattern du E111. Faute de place, les valeurs trop longues ont été tronquées. La reconnaissance n'a néanmoins pas souffert.

Malgré le fait que cette méthode semble marcher, il n'y a pas de quoi crier victoire. La reconnaissance a un prix non négligeable : la taille des patterns d'entrée. Il n'est pas exceptionnel d'avoir plus de 50 intersections d'un seul type. Une possibilité est alors de diviser l'entrée par une constante, mais évidemment la reconnaissance y perd.

III.2.6. CONCLUSIONS

Dans l'ensemble l'espoir de réaliser la reconnaissance de formulaires par Perceptron est mince.

Si nous l'essayons immédiatement sur le bitmap, toute déformation par translation, rotation ou modification d'échelle interdit toute reconnaissance significative.

Nous avons utilisé ensuite les caractéristiques déjà calculées dans le cadre de la méthode classique, et nous avons greffé le réseau comme base de données à accès rapide sur le système de reconnaissance. De nouveau, deux possibilités se présentaient : introduction des données en base dix ou introduction de données binaires.

Dans le premier cas, la reconnaissance ne se faisait pas.

Dans le dernier cas, la seule représentation trouvée qui semble donner des résultats valables, n'utilise pas l'espace mémoire de façon efficace.

Le résultat apporté par un réseau Perceptron est, dans pas mal de cas, loin d'être clair. Pour trancher la question, utiliser une fonction de sortie à seuil est un peu brutal. Il en résulte que toute implémentation nécessite un interpréteur de résultats.

Pour avoir une idée intuitive sur le comportement du Perceptron, nous nous sommes basés sur l'heuristique de "ressemblance", la ressemblance étant définie par le nombre de neurones que deux patterns ont en commun. Le pattern appris qui ressemble le plus au pattern introduit sera très probablement le résultat fourni par le réseau.

Le Perceptron n'apporte pas de solution satisfaisante. Que pouvons-nous attendre des réseaux récurrents, aussi connus sous le nom de réseaux de Hopfield?

III.3. Le réseau Hopfield.

III.3.1. INTRODUCTION.

Un réseau de Hopfield est un réseau récurrent à une seule couche dont la conception principale est la satisfaction de contraintes. Comment utiliser ce type de réseau dans la reconnaissance de formulaires?

Dans la reconnaissance d'objets 2-D ou 3-D par réseau de relaxation, la tendance générale est :
- d'extraire les caractéristiques des objets à reconnaître,
- d'en faire un graphe,
- de le comparer, par réseaux neuronaux, aux graphes des modèles stockés.

[Parvin, 1989] ou [Tresp, 1990], par exemple, suivent cette tendance.

La reconnaissance se fait par correspondance entre graphes ("sub-graph matching"). Un réseau de type Hopfield se prête bien à ce type de matching, car les contraintes expriment deux choses principales :
- une correspondance 1 à 1 ou 1 à 0
- la distance minimale entre des valeurs d'attributs.

Les attributs sont spécifiques au problème traité et la correspondance peut être reprise par des contraintes dans les poids et/ou par des contraintes dans le calcul de l'énergie du réseau.

La méthode prévoit pour chaque modèle de la base de données, un réseau différent. La base de données croît linéairement avec le nombre de modèles.

III.3.2. DESCRIPTION DU RÉSEAU.

Comment utiliser les caractéristiques calculées du formulaire dans un réseau Hopfield? Comme nous voulons garder un réseau "standard" de Hopfield, les contraintes sont simples. Etant donné la nature des

caractéristiques, une comparaison entre le nombre de caractéristiques dans le formulaire et un modèle est effectuée.

Le nombre d'intersections d'un type est indiqué en mettant une entrée correspondante à un. Par exemple, s'il y a 3 intersections de type "+", la ligne correspondante au type "+" dans la matrice d'entrée aura la forme (1 0 0) .

La matrice des poids n'est pas calculée par le réseau comme dans le Perceptron. Dans les réseaux Hopfield, elle est construite par l'utilisateur. Elle reflète les contraintes. Les exemples que nous utilisons pour le Hopfield ne sont pas pratiques car une matrice de 81^2 poids, soit 6561, doit être utilisée. Nous nous limitons donc à un exemple de 2 lignes avec 2 valeurs possibles. L'idée principale reste intacte.

Nous avons en entrée 8 neurones (figure 3-21). Les 4 neurones de début sont attribués au formulaire réel; les quatre dernières appartiennent au modèle. La couche principale du réseau comprend également 8 neurones. Aucune optimisation du réseau n'a été recherchée, nous ne voulions examiner uniquement la faisabilité. La clarté reste plus importante que la performance.

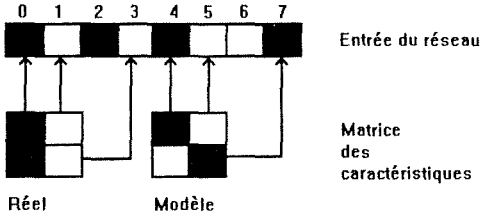

Figure 3-21 : Entrée du réseau de Hopfield.

III.3.3. ALGORITHME DE HOPFIELD.

De façon générale l'algorithme prend la forme suivante :

Initialiser la température T.
NCYCLES fois :
  NNEURONES fois :
    Choisir aléatoirement un neurone i,
    Calculer son niveau d'activation a_i,
    Si pr(a_i) < Rnd, alors a_i = 1 sinon a_i = 0.
  Calculer le Niveau_Energie du réseau dans son état courant,
  Réduire la température T.

où **NCYCLES** est une constante indiquant le nombre de cycles à parcourir,

NNEURONES est une constante : le nombre de neurones qu'il faut modifier par cycle,

Rnd est une valeur aléatoire dans l'intervalle [0, 1[.

III.3.4. IMPLÉMENTATION DU RÉSEAU.

Une implémentation des mêmes auteurs [Rumelhart-3, 1988] est choisie. Elle permet au réseau récurrent de travailler en plusieurs modes. Nous utiliserons le mode Hopfield "simple" et le mode Boltzmann. Le mode Hopfield n'inclut pas de recuit simulé et est formé de neurones à valeurs continues. Un réseau de Boltzmann, plus souvent appelé "machine de Boltzmann", contient des neurones binaires et est soumis au recuit simulé.

III.3.4.1. Description du réseau.

```
définitions:
nunits 8
ninputs 8
nupdates 8
end
network:
%. 0 8 0 8
end
```

```
biases:
%. 0 8
end
```

Figure 3-22 : Description du réseau Hopfield utilisé.

Dans les *définitions* on décrit que le réseau contient 8 unités en tout, sans les neurones d'entrée; qu'il y a 8 éléments d'entrée et que par cycle neuf neurones doivent être mis-à-jour.

Dans la partie *network* nous trouvons 64 connexions : chaque neurone est lié à chaque autre neurone.

Les connexions et les "tendances" (biases) sont déclarées immuables. Leurs valeurs seront lues dans le fichier des poids dont nous parlerons plus loin.

III.3.4.2. Paramètres du réseau.

```
get network hopf.net
get weights hopf.wts
set ncyc 20
set param estr .4
set param istr .4
set mode boltz 1
get anneal 2 20 .05 end
```

Figure 3-23 : Paramètres du réseau.

D'abord deux fichiers sont lus : la description du réseau et les poids des connexions. Ensuite le nombre de cycles à parcourir est défini. Les paramètres *estr* et *istr* correspondent à l'importance que l'on donne à chacun des termes de la fonction d'entrée d'un neurone (voir II.5.1.) Les valeurs mises sont des valeurs empiriques. Ensuite, nous indiquons que nous souhaitons travailler en mode Boltzmann. Et finalement, les paramètres du recuit simulé sont mis : d'abord une température initiale (le 2), puis une paire "temps-température". Ici, nous voulons qu'au cycle 20 la température ne soit plus que de 0,05. La diminution à chaque cycle est calculée par simple interpolation.

III.3.4.3. Les poids des connexions.

Comme nous l'avons annoncé dans l'introduction, nous voulons créer un réseau avec des contraintes simples. Et les contraintes sont liés aux nombres de caractéristiques des différents types.

Supposons que nous n'avons que deux caractéristiques et pour chacune d'elles nous avons 0 ou 1 occurrence possible. Nous pouvons donc construire la matrice suivante :

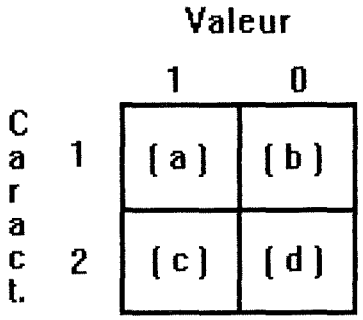

Figure 3-24 : Matrice de caractéristiques.

Supposons que la caractéristique 1 apparaît sur le formulaire, alors la case (a) est mise à 1 et la case (b) à 0. Si la caractéristique 2 est absente, le (c) contient un 0 et (d) un 1.

Nous trouvons ici une première contrainte à intégrer dans le réseau : chaque ligne de la matrice ne peut contenir au maximum qu'un seul 1. (Contrainte 1)

Dans l'input du réseau nous couplons les matrices de deux formulaires : le formulaire à reconnaître et un formulaire de référence. Voici les huit neurones d'entrée. Plus le formulaire modèle ressemble au formulaire réel, plus nous voulons que l'énergie du système soit élevée. Toute correspondance entre les deux matrices doit être récompensée, toute différence découragée. (Contrainte 2)

Rappelons qu'il y a deux contraintes supplémentaires sur l'attribution des poids de connexion qui garantissent la convergence. La matrice doit être symétrique et les éléments de la diagonale principale doivent être égal à 0.

```
 0 -1   0  0      2 -1   0  0
-1  0   0  0     -1  2   0  0

 0  0   0 -1      0  0   2 -1
 0  0  -1  0      0  0  -1  2

 2 -1   0  0      0 -1   0  0
-1  2   0  0     -1  0   0  0

 0  0   2 -1      0  0   0 -1
 0  0  -1  2      0  0  -1  0

 1  1  1  1  1  1  1  1
```

Figure 3-25 : Poids des connexions.

La dernière ligne sont les "tendances" (biases). Ils sont mis à 1 pour éviter toute influence du 0 comme élément absorbant de la multiplication. Les autres valeurs sont les poids des connexions.

La figure 3-25 est divisé en quatre, et puis à nouveau en quatre, pour la lisibilité. Chacun des quatre quadrants forme une unité sémantique.

Les quadrants 1 et 4 appliquent la contrainte 1. Par exemple, si nous prenons le premier quadrant dans le quadrant 1, nous trouvons 0 -1, -1 0. Les deux occurrences de -1 doivent être comprises comme suit : dans la première ligne de la matrice des caractéristiques, ne peuvent apparaître que 0 0, 1 0 ou 0 1. La combinaison 1 1 fait décroître l'énergie. Toute incohérence dans les entrées peut être signalée ainsi par le réseau. Il suffit de poser le poids d'inhibition à une valeur prohibitive.

Les quadrants 2 et 3 sont la contrainte 2. Analysons le quadrant 2. Il doit de nouveau être divisé en quatre sous-quadrants. La première ligne du sous-quadrant 1 (2 -1) décrit les connexions entre la première ligne de la matrice des caractéristiques du formulaire réel et la première ligne du modèle. Si le premier neurone est actif dans les deux cas, une récompense de 2 est attribuée pour la correspondance. S'il y a discordance (1 0 et 0 1), une "punition" (inhibition) de -1 est attribuée. Pour accentuer la différence qu'il y a, on met le poids négatif en correspondance avec la distance entre les deux neurones actifs. Par exemple, (1 0 0) comparé à (0 0 1) donne un poids de -2. Ainsi l'énergie reflète la distance totale entre le nombre de caractéristiques du formulaire et celui du modèle. Plus elle est élevée, mieux c'est.

Les sous-quadrants 2 et 3 de tous les quadrants montrent les liens entre la première et la seconde ligne des matrices. Dans notre exemple, les caractéristiques sont indépendantes et donc tous ces liens sont égal à 0.

III.3.5. RÉSULTATS.

Nous procédons à trois comparaisons.

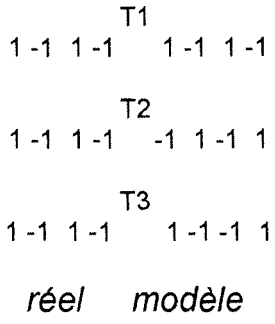

Figure 3-26 : Entrées du réseau Hopfield.

Nous comparons trois modèles différents au même formulaire réel.

Entrée	*Energie*
T1	4.8000
T2	3.2000
T3	4.0000

Figure 3-27 : Résultats du réseau Hopfield.

T1 est le cas où la correspondance est parfaite. Le niveau d'énergie résultant est le plus grand. Dans le T3, la différence est d'un seul élément actif. Le T2 correspond à un modèle de la base de données qui est complètement différent du formulaire réel. Les résultats correspondent à ce à quoi nous nous attendions. Le principe reste inchangé si nous appliquons cette méthode à des matrices de caractéristiques plus grandes : comparer tous les modèles au formulaire réel et garder la meilleure correspondance.

Les résultats viennent d'un réseau en mode Boltzmann. Le mode Hopfield simple donne exactement les mêmes résultats. Ceci parce que la fonction d'énergie est relativement simple et ne possède pas de maxima locaux assez importants pour influencer l'algorithme de relaxation. Trouver le maximum prend, en mode Hopfield, 9.5 cycles en moyenne. En mode Boltzmann, la réponse prend 16.3 cycles.

Il serait faux d'en conclure que le mode Hopfield est meilleur que le mode Boltzmann pour des configurations plus grandes et complexes. Tout dépend de la forme de la fonction d'énergie pour le réseau. S'il s'agit d'une colline, le mode Hopfield est le plus adéquat. Si, au contraire, la fonction d'énergie a la forme de l'Himalaya, le mode Hopfield grimpe la première montagne venue et s'installe au sommet. Le mode Boltzmann, descend de temps en temps pour escalader une montagne différente. Plus cette montagne est haute, plus la probabilité est grande qu'elle devienne le résultat final. Il est donc évident que le temps de réponse est plus long, mais il y a de fortes chances que le résultat soit le Mont Everest.

III.3.6. CONCLUSIONS.

Nous avons proposé une méthode qui satisfait à la reconnaissance par l'utilisation des caractéristiques du formulaire. Elle est réalisable mais loin d'être efficace.

Premièrement, il y a la question de la représentation du nombre de caractéristiques par des matrices décrites au point III.3.2. Si, par exemple, nous prenons pour chacune des 9 caractéristiques 50 colonnes pour un seul formulaire, on obtient 50 * 9 = 450 neurones.

Secundo, la méthode utilise le formulaire et un modèle dans le même réseau, d'où la nécessité de 900 neurones. L'interconnexion complète : 810 000 connexions, pour lesquels il faut créer un fichier des poids. Le fichier est générable automatiquement grâce à sa structure.

Ne parlons pas du temps calcul, et donc du temps de réponse. Il y a 2^{900} états possibles du réseau.

Troisièmement, chaque réseau ne compare qu'un seul modèle au formulaire. Donc, la base de données croît linéairement avec le nombre de formulaires de référence. Il est vrai que les comparaisons peuvent se faire en parallèle, mais ce n'est pas vraiment une consolation.

Nous pouvons constituer une liste des modèles, ordonnés sur la ressemblance, ce qui est un avantage.

En conclusion, la méthode proposée convient pour des petits réseaux, mais est tout à fait impossible à la lumière des hypothèses de travail que nous nous sommes données pour la reconnaissance de formulaires.

RESEAU NEURONAL D'ORDRE SUPERIEUR

IV.1. Introduction.

Si les réseaux de type Perceptron faillissent en présence de transformations, c'est dû au fait que l'influence des pixels est purement individuelle. Il n'y a pas de relation définie entre les éléments du bitmap. Les réseaux d'ordre supérieur permettent d'y remédier.

Le réseau implémenté dans ce mémoire est un Perceptron d'ordre 2 à une seule couche. Il s'agit d'un réseau de 49 noeuds d'entrée (bitmap de 7 * 7 pixels) et 4 neurones de sortie. Les noeuds d'entrée sont binaires; les noeuds de sortie, à valeurs continues.

Les interconnexions entre neurones sont feed-forward, et uniquement d'ordre deux[13]. (Voir figure 4-1.) Nous parlons donc d'une combinaison de 2 éléments parmi 49 à savoir 1176 connexions (contre 49 * 4 = 196 dans le Perceptron d'ordre 1).

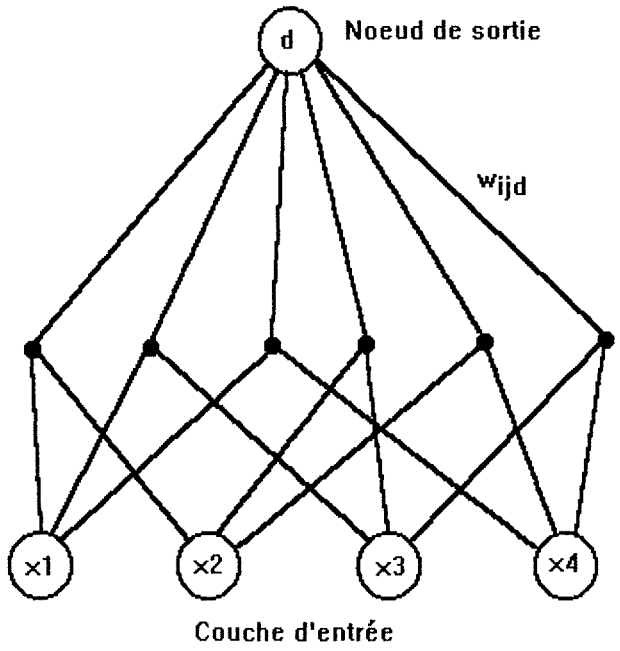

Figure 4-1 : Réseau avec uniquement des connexions d'ordre 2.

[13] Dans la formule générale de la fonction d'activation du réseau d'ordre supérieur nous retenons uniquement le second terme, en forçant toutes les autres connexions à 0.

Le réseau n'a qu'une seule couche et donc l'apprentissage du réseau peut se faire par une simple règle delta qui garantit une convergence rapide [Rosenblatt, 1962].

L'invariance à l'échelle et à la translation est implémentée sous forme de liens entre les éléments d'entrée qui ont la même inclinaison pour la droite définie par les deux points. Toutes les connexions liées ont le même poids. Nous devons à ce sujet répondre à deux questions:

(a) comment implémenter le lien entre les poids et
(b) comment déterminer le poids unique?

En ce qui concerne la partie (b), le choix est tombé sur la moyenne arithmétique.

La question (a) demande une préparation d'une matrice des liens. Cette préparation se fait par un programme séparé dont le détail algorithmique se trouve dans l'annexe B. L'algorithme est divisé en deux parties :

1) **Pour toute paire d'éléments du bitmap d'entrée :**

   ((p1,p2) avec p2 ≥ p1)

   Calculer l'inclinaison de la droite p1p2

   par la formule (y2-y1)/(x2-x1)

   où p1=(x1,y1) et p2=(x2,y2).

   Sauver cette valeur dans $L_{p1,p2}$

   avec L la matrice des liens.

2) **Relier, par pointeurs, tous les éléments $L_{i,j}$ avec la même valeur en une chaîne fermée.**

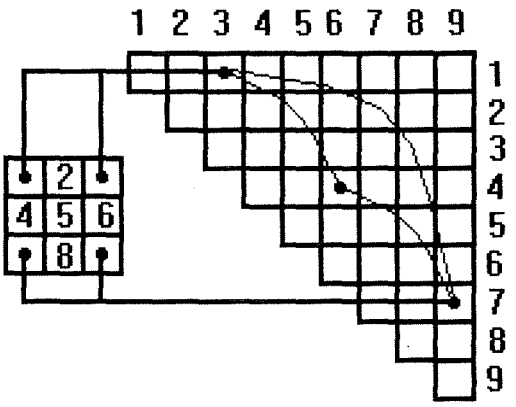

Figure 4-2 : Matrice des liens.

Comme ce traitement est indépendant des réseaux neuronaux, il sauvegarde ses résultats sur fichier.

IV.2. Algorithme du réseau d'ordre supérieur.

IV.2.1. SPÉCIFICATIONS.

IV.2.1.1. Réseau d'ordre supérieur.

Arguments : W: matrice des poids,
MNinput: matrice des neurones d'entrée,
MNoutput: matrice des neurones de sortie,
MNtrain: matrice du pattern à apprendre,
Apprentissage: drapeau du mode d'apprentissage,
Erreur_acceptable: seuil d'arrêt d'apprentissage.

Préconditions :
Apprentissage = "VRAI" ou "FAUX".

Résultats : W ',
MNoutput'.

Postconditions :
{ (Apprentissage = "FAUX")
∧
(MNoutput' = MNoutput
avec $\forall$ $a_d \in$ MNoutput : a_d = logistique($\Sigma_i \Sigma_j$ w_{ijd} y_i y_j))
} ∨

$$
\begin{aligned}
\{\ & (\text{Apprentissage} = \text{"VRAI"}) \\
& \wedge \\
& (\text{MNoutput}' = \text{MNoutput} \\
& \qquad \text{avec } \forall\, a_d \in \text{MNoutput} : \\
& \qquad\qquad a_d = \text{logistique}(\Sigma_i \Sigma_j\, w_{ijd}\, y_i\, y_j)) \\
& \wedge \\
& (W' \neq W \\
& \qquad \text{avec } w'_{ijd} = w'_{kld} \\
& \qquad\qquad \forall\, i, j, d, k, l \,/\, j>i,\ k>l : (j_y - i_y)/(j_x - i_x) = \\
& \qquad\qquad\qquad (l_y - k_y)/(l_x - k_x)) \\
& \wedge \\
& (\Sigma_d\, (\text{MNtrain}_d - \text{MNoutput}_d) < \text{Erreur\_acceptable}) \\
\}\ &
\end{aligned}
$$

IV.2.1.2. Correction des poids de connexion.

Arguments : W : matrice carrée (supérieure) des poids.
Préconditions :

Résultats : W '
Postconditions :

$$
\begin{aligned}
& W' \neq W \text{ avec} \\
& \qquad \forall\, i', j', d' \ / j' > i' \ : \ w'_{i'j'd'} = \Sigma w_{ijd} \,/\, N \\
& \qquad\qquad \forall\, i, j \,/\, j > i : (j_y - i_y)/(j_x - i_x) = (j'_y - i'_y)/(j'_x - i'_x)
\end{aligned}
$$

IV.2.2. ALGORITHME.

IV.2.2.1. Initialisations.

```
NINPUT ← 49.                 { Le nombre de noeuds d'entrée.}
NOUTPUT ← 4.                 { Le nombre de neurones de sortie.}
ERREUR_ACCEPTABLE ← 0.1      { Le seuil de la réussite d'apprentissage.}
Lire en entrée : NPATTERNS,  { Le nombre de patterns à traiter.}
                 MODE_APP,   { Drapeau du mode "apprentissage".}
                 NCYCLES.    { Le nombre maximal de cycles
                               d'apprentissage. }
```

IV.2.2.2. Corps du traitement.

Si MODE_APP = FAUX

Alors { Uniquement résultat du matching.}

Lire le fichier contenant le pattern à reconnaître.

Lire le fichier de poids de connexions.

Pour chaque neurone de la couche de sortie :

Calculer le niveau d'activation du neurone selon la formule d'ordre 2 :

$a_d \leftarrow$ **logistique(** $\Sigma_i \Sigma_j w_{ijd} * y_i * y_j$ **).**

avec : a_d : le niveau d'activation du neurone d,
w_{ijd} : poids de la connexion d'ordre 2 de i et j vers d,
y_i et y_j : sorties des neurones de la couche d'entrée,
logistique : appliquer $f(x) = 1/(1+e^{-x})$.

Afficher les valeurs des neurones de sortie.

Sinon { Apprentissage des patterns. }

Les poids des connexions sont initialisés à un nombre aléatoire entre 0 et 1.

Lire le fichier des liens.

Corriger les poids liés.

NCYCLES fois :

Erreur_sur_cycle = 0,

NPATTERNS fois :

Lire le fichier du pattern à apprendre.

Lire le fichier du pattern de sortie correspondant.

Calculer la sortie comme auparavant.

Calculer l'erreur pour ce pattern par la formule :

Erreur_pattern = $\Sigma_d (t_d - y_d)^2$

où t_d est le $d^{ième}$ élément du pattern attendu et
y_d est le neurone de sortie d.

Erreur_sur_cycle = Erreur_sur_cycle + Erreur_pattern.

Corriger les poids des connexions. (Voir II.6.4.)

Corriger les poids liés.

Si Erreur_sur_cycle < ERREUR_ACCEPTABLE,

alors sauver les poids des connexions dans un fichier,

s'arrêter.

La procédure de correction des poids de connexion :

Pour toute chaîne de poids liés :

Sommer toutes les valeurs des éléments de la chaîne.

w_{ijd} <- somme / nombre_d'éléments.

IV.3. Résultats.

IV.3.1. PATTERNS DE RÉFÉRENCE.

Nous présentons au réseau quatre patterns : une ligne verticale, une horizontale et deux diagonales.

L'apprentissage s'est fait en seulement trois cycles. Ceci est dû au nombre de connexions que le réseau a à sa disposition pour distribuer la connaissance.

1100000	0000000	0000011	1100000
1100000	0000000	0000110	0110000
1100000	1111111	0001100	0011000
1100000	1111111	0011000	0001100
1100000	0000000	0110000	0000110
1100000	0000000	1100000	0000011
1100000	0000000	1000000	0000001
1000	0100	0010	0001

Figure 4-3 : Patterns de référence.

IV.3.2. PATTERNS DE RECONNAISSANCE ET RÉSULTATS.

0000011	0000000	0010000	0000000	0000000
0000011	0000000	0100000	0000000	0000000
0000011	0000000	1000000	0001000	0001000
0000011	0000000	0000000	0001000	0011100
0000011	0000000	0000000	0001000	0001000
0000011	1111000	0000000	0000000	0000000
0000011	0000000	0000000	0000000	0000000
Test1	Test2	Test3	Test4	Test5

Figure 4-4 : Patterns à reconnaître.

Résultat du Test1 : **1.00 0.00 0.00 0.00**

Il s'agit d'une simple translation de la verticale; aucun changement d'échelle. La reconnaissance est parfaite. Les valeurs des neurones de sortie ne sont pas soumises à une fonction de seuil. Ce qui rend le résultat d'autant plus remarquable.

Ici, le pas suivant est d'analyser les poids des connexions et d'essayer de déduire le degré de translation à partir des valeurs.

Résultat du Test2 : **0.05 0.37 0.03 0.02**

Il y a eu translation et modification d'échelle. Apparemment, l'échelle a beaucoup influencé le degré de "certitude". Mais malgré que le 0.37 est loin de 1, relativement parlant cette valeur se distingue clairement et la bonne réponse est de nouveau trouvée.

Résultat du Test3 : **0.07 0.06 0.16 0.04**

Résultat du Test4 : **0.20 0.06 0.04 0.03**

Dans les deux cas nous pouvons constater les mêmes choses que pour le Test2. Les résultats de Test2, Test3 et Test4 laissent ressentir qu'il y a une relation entre les patterns Test et son pattern de référence. Or il n'existe pas de corrélation linéaire ni logarithmique satisfaisante.

Résultat du Test5 : **0.02 0.02 0.01 0.00**

Le Test5 nous donne une agréable surprise : le réseau ne choisit aucun pattern de référence. Il n'en choisit pas un "au hasard".

De cette série de tests nous pouvons conclure deux choses intéressantes : si nous utilisons une fonction de sortie à seuil, toutes les bonnes réponses ont été trouvées. Le seuil à choisir pour ce réseau est de 0.10. D'autre part, toute approche de représentation de résultat doit prévoir un pattern correspondant à "INCONNU".

A titre de comparaison, les mêmes patterns de référence et de tests ont été utilisés sur le même réseau, mais celui-ci, cette fois, ne tient pas compte des invariances, c'est-à-dire des liens entre les

connexions. Ceci réduit le réseau à un Perceptron avec une fonction d'activation complexe. Le réseau nécessite un seul cycle pour apprendre les quatre patterns de référence.

Pattern					
Test1	:	1.00	1.00	1.00	1.00
Test2	:	0.95	0.37	0.96	0.32
Test3	:	0.66	0.24	0.08	0.14
Test4	:	0.00	0.06	0.18	0.08
Test5	:	0.00	0.01	0.00	0.00

De cet apprentissage ultra-rapide et de ce résultat nous ne savons pas conclure grand chose. Mais le point est fait : l'invariance est bel et bien la cause de la reconnaissance des patterns.

IV.4. Conclusions.

Il est clair que l'invariance, du moins par rapport à l'échelle et à la translation, est réalisable par ce type de réseau.

De plus, la convergence pendant la période d'apprentissage est rapide.

L'inconvénient est évidemment l'explosion combinatoire des connexions. Cette explosion influence de façon directe le temps de réponse et l'espace mémoire nécessaire. Voici quelques chiffres :

Pattern	ordre 2	ordre 3
7 * 7	1176	18424
10*10	4950	161700
16*16	32640	2763520
32 * 32	523776	1.784 E 8

Une page A4 contient, en 300 dpi[14], 3425 * 2563 = 8778275 pixels.

14 300 dpi (dots per inch) est une résolution standard en matière de d'images.

LA METHODE CLASSIQUE.

La méthode classique est ainsi appelée pour l'opposer à la méthode par réseau neuronal. Ce chapitre explique la reconnaissance du formulaire et le calcul des paramètres de transformation réalisés dans le cadre du système d'analyse de formulaires chez Philips. Les chapitres I et V du mémoire sont des résumés de mon travail au centre de recherche. Plus de détails peuvent être trouvés dans [Philips-1, 1990].

V.1. Retrouver le formulaire dans la base de données.

V.1.1. INTRODUCTION.

Comme les caractéristiques de tous les formulaires que nous voulons reconnaître se retrouvent dans la base de données, il suffit de les comparer aux caractéristiques du formulaire en cours. Les frames de la base de données peuvent alors être triés sur ressemblance et le premier de la liste est considéré comme le candidat le plus probable. Cette ressemblance est calculée par la somme des différences entre les caractéristiques calculées pour le formulaire courant et chacun des modèles.

V.1.2. SPÉCIFICATIONS.

Arguments :

1) B : base de données des modèles, contenant des frames simplifiés avec caractéristiques.
2) F : frame simplifié avec caractéristiques.

Préconditions :

$B = \{M_j\}$, non-vide.

Résultats :

B' : base de données contenant des frames simplifiés avec caractéristiques.

Postconditions :

B'= B, dont les éléments sont ordonnés, en croissant, sur la différence entre M_i et F.

La différence = Σ (abs(M_{ic}-F_c))

avec M_{ic} = caractéristique c du frame M_i,

et F_c = caractéristique c du frame F.

V.1.3. EXEMPLE.

Nous avons constitué une base de données de cinq formulaires qui portent les noms poétiques : F11, F21, F41, F51 et F61.

F11	0	:	26
F41	90	:	34
F41	270	:	34
F41	180	:	36
F41	0	:	36
F11	90	:	38
F11	180	:	40
F11	270	:	40
F21	0	:	67
F21	180	:	67
F21	90	:	71
F21	270	:	71
F61	0	:	74
F61	270	:	78
F51	180	:	79
F51	270	:	79
F51	0	:	79
F51	90	:	79
F61	180	:	84
F61	90	:	84

Figure 5-1 : la liste triée des candidats pour le formulaire E111.

V.2. Le calcul des paramètres de transformation.

Cette deuxième partie du chapitre propose une solution au deuxième volet des spécifications de sortie du système de reconnaissance de formulaires (cf. I.2.1.) Il existe plusieurs méthodes pour retrouver les paramètres de transformation.

V.2.1. CORRESPONDANCE DE POINTS.

Il faut pour cette méthode pouvoir déterminer avec certitude un nombre de points du formulaire courant dans le modèle courant. Des outils mathématiques informatisés existent pour calculer les paramètres de transformation. Tout le problème réside évidemment dans la recherche de points correspondants.

V.2.2. MÉTHODES DE TRANSFORMATION.

L'idée est d'incrémenter des accumulateurs, en fonction des correspondances, dans un espace multidimensionnel et ensuite d'utiliser comme résultat les valeurs maximales. Elle est connue sous le nom de "Hough transforms".

J'ai implémenté cette méthode. Par raffinement successif sur la précision, j'ai essayé de trouver des paramètres de translation et d'échelle. Pour chaque paire de lignes un accumulateur, correspondant à la transformation à appliquer, est incrémenté selon la longueur de la ligne. Les lignes longues sont plus importantes que les courtes.

Cette méthode fonctionne bien tant que le formulaire est sans bruits. Les problèmes commencent quand, par le bruit, les longues lignes sont coupées. Malheureusement, il en est trop souvent ainsi pour que le système fonctionne de façon satisfaisante.

Mes résultats ont été confirmés par les travaux dans le même domaine. Par exemple, Grimson [Grimson,1990] a trouvé par expérimentation les limites suivantes:

1. La méthode ne fonctionne pas bien en dans les cas de modification d'échelle.
2. Elle n'est pas fiable en la présence d'un environnement complexe, par exemple de multiples objets et occlusion d'objets.
3. Elle produit un nombre très élevé de résultats possibles quand les images sont bruités.

V.2.3. ALGORITHMES DE PARCOURS D'ARBRE.

La méthode "Hough transforms" se base sur la supposition que la majorité des lignes du frame courant correspondent à leurs équivalents dans le modèle. Dans l'optique suivie, nous supposons qu'au moins une paire de lignes nécessite les paramètres de transformation corrects ou

très approximatifs. Partons d'abord du point de vue qu'aucune modification d'échelle n'a eu lieu. Si cette hypothèse ne permet pas de trouver un modèle satisfaisant, nous avons le choix entre un nombre élevé de méthodes très connues de parcours d'arbre pour arriver à trouver cette paire. La méthode d'appréciation de la correspondance entre les deux frames est plus complexe.

V.2.3.1. Recherche des paramètres de transformation sans modification d'échelle.

Les cas de routine sont des formulaires qui n'ont subi que peu de transformations. Il arrive qu'une photocopieuse réduit un formulaire d'un ou de deux pour cent, mais dans la grande majorité des cas le formulaire est à une échelle de cent pour cent. Trouver dans ce cas les paramètres de translation est une opération rapide: il suffit de faire glisser le frame courant sur le frame du modèle dans chaque direction. Une mesure de confiance de correspondance entre les deux frames est calculée par comparaison des lignes.

Spécifications :

Arguments :

M : Liste de modèles.
F : Frame.
S : Seuil d'acceptation.

Préconditions :

M est ordonné, croissant, sur la différence des M_i avec F.
F est un frame simplifié avec caractéristiques.

Résultats :

M_K : modèle appartenant à M,
θ : paramètres de transformation.
NULL : résultat vide.

Postconditions :

M_K si $Match(M_K, F, \theta) \geq S \wedge$ échelle(θ) = 1
où $Match(M_K, F, \theta)$ est la mesure de correspondance entre le modèle M_K et le frame F avec les paramètres de transformation θ. (Voir V.2.3.3).
NULL sinon.

Recherche des paramètres de translation

Sommer la longueur des lignes colinéaires, avec une certaine tolérance, dans un vecteur. Ce vecteur des lignes représente une sorte

d'histogramme pour ce formulaire (voir figure 5-2). La même chose est faite pour chaque modèle et le frame courant.

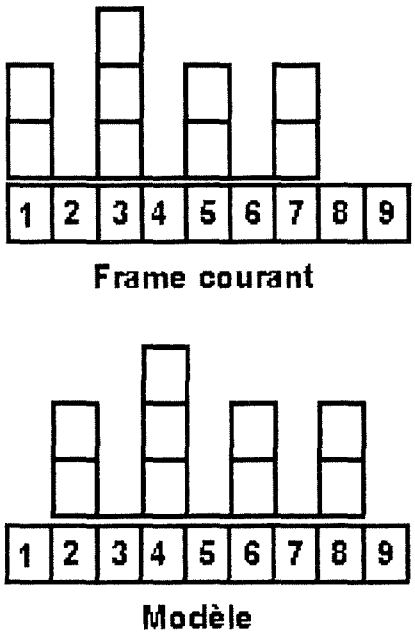

Figure 5-2 : exemple de vecteurs des lignes.

En cherchant le déplacement nécessaire pour avoir une différence minimale entre les deux vecteurs des lignes, nous obtenons la translation voulue. Dans la figure 5-2, nous constatons une translation de +1.

V.2.3.2. Recherche des paramètres de transformation avec modification d'échelle.

Si la mesure de confiance reste en dessous d'un seuil, une recherche plus longue, en tenant compte d'une modification d'échelle, est effectuée. Parmi les nombreuses possibilités de parcours d'arbre, un "branch and bound" simple est choisi. Si des valeurs de transformation sont trouvées pour lesquelles la mesure de confiance vis-à-vis de la correspondance dépasse un seuil donné, elles sont acceptées. Sinon, la recherche produit comme résultat la valeur "formulaire inconnu" et l'intervention d'un opérateur humain est nécessaire.

Spécifications :

Arguments :

M : Liste de modèles.
F : Frame.
S : Seuil d'acceptation.
T : Seuil de sélection.

Préconditions :

M est ordonné, croissant, sur la différence des M_i avec F.
F est un frame simplifié avec caractéristiques.

Résultats :

M_k : modèle appartenant à M,
θ : paramètres de transformation,
"Formulaire inconnu".

Postconditions :

M_k si ($Match(M_k, F, \theta) \geq S \wedge 0.70 \leq échelle(\theta) < 1.40$),
$\{M_k\}$ si ($T \leq Match(M_k, F, \theta) < S \wedge 0.70 \leq échelle(\theta) < 1.40$),
"Formulaire inconnu" sinon.

où $Match(M_k, F, \theta)$ est la mesure de correspondance entre le modèle M_k, le frame F avec les paramètres de transformation θ. (Voir V.2.3.3).

Remarque

Il serait agréable de pouvoir reconnaître des formulaires de n'importe quelle taille. Malheureusement, ceci est rendu impossible par la routine d'extraction des lignes à partir du bitmap. Nous en profitons pour optimiser la recherche décrite dans cette section, car toute échelle prohibitive est immédiatement éliminée.

V.2.3.3. La mesure de correspondance entre deux frames.

Cette procédure chiffre la ressemblance entre deux frames en les comparant ligne à ligne.

Spécifications :

Arguments :

θ : paramètres de transformation.
F_1: frame.
F_2: frame.

Préconditions :

$\theta = (tx, ty, \varepsilon, \alpha)$,

où : tx = translation horizontale,
ty = translation verticale,
ε = facteur d'échelle,
α = rotation à appliquer sur F_1 pour arriver à F_2

Résultats :

Match(F_1, F_2, θ) = mesure de correspondance entre le frame F_1 et le frame F_2 en appliquant la transformation θ sur F_1.

Postconditions :

$$\text{Match}(F_1, F_2, \theta) = \frac{\sum_k \text{len}(S_k F_1)}{\sum_j \text{len}(L_j F_1)}$$

où $\text{len}(S_k F_1)$ = longueur du segment de ligne de $L_i F_1$
tel que $S_k F_1 = (L_i F_1 \cap L_i F_2)$

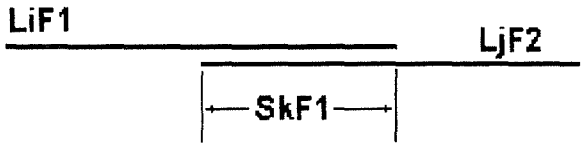

Figure 5-3.

V.2.3.4. Exemple.

```
E111   0 : 0.00 0.00 --> 88
```

Figure 5-4 : la liste des transformations proposées et la mesure de confiance.

Dans cet exemple, le formulaire E111, avec une rotation de 0 degré et avec des translations horizontales et verticales de 0 cm, est proposé comme le meilleur candidat avec une correspondance de 88 % entre le formulaire rempli et le modèle de la base de données. Cette valeur dépasse le seuil de 80%, paramètre du système, et elle est donc immédiatement acceptée comme résultat.

V.3. Les résultats.

Sur une machine PC/386 avec coprocesseur mathématique, le temps de réponse est immédiat. En moins de 1 seconde le programme rend la solution :

référence : E111
sca : 1.00
tra x & y : 0.00 0.00
x, y & a : 10.50 14.85 0.00

Figure 5-5 : Résultat de la reconnaissance par une méthode non-neuronale.

Nous devons comprendre cela comme suit : le modèle reconnu est le E111 avec une échelle de 1, des translations de 0 centimètre et une rotation, autour de l'axe (10.5, 14.85), de 0 degré.

Si le seuil de 80% n'est pas dépassé par un des modèles, lors de la vérification de correspondance, la méthode exhaustive est lancée. Si nous demandons la reconnaissance d'un formulaire non-existant dans la base, sur la même machine, la réponse "FORMULAIRE INCONNU" apparaît après 90 à 100 minutes seulement.

L'apprentissage d'un modèle est encore plus rapide que la reconnaissance. Après le pré-traitement, les caractéristiques calculées et le frame sont ajoutés à la base des formulaires. La seule chose à laquelle il faut faire attention lors de la création de la base des modèles, est d'utiliser des modèles vierges et sans transformation.

L'espace mémoire utilisé est évidemment linéairement dépendant du nombre de modèles à reconnaître. Selon le nombre de lignes sur le formulaire, il faut compter entre 2 Kbytes et 4 Kbytes d'espace disque par modèle à reconnaître.

CONCLUSIONS ET RECHERCHES ULTERIEURES

VI.1. Introduction.

Les résultats donnés dans l'ensemble du mémoire le sont toujours à titre indicatifs. Les réseaux neuronaux n'ont jamais été optimisés. Cette optimisation doit se manifester par un choix judicieux du nombre de neurones dans chaque couche et par le choix des paramètres. Les choix ont été fait dans le but de démontrer la capacité ou l'incapacité des réseaux à satisfaire à un besoin particulier.

VI.2. Comparaison des méthodes.

Si nous parcourons les trois réseaux neuronaux et la méthode classique, que pouvons nous conclure?

	Avantages	Inconvénients
PERCEPTRON		
Reconnaissance de bitmaps		- pas de reconnaissance si transformation du formulaire réel.
Reconnaissance par caractéristiques en base dix		- pas de reconnaissance
Reconnaissance par caractéristiques en binaires	- espace mém. const. - temps de réponse constant	- espace mémoire grand; - interprétation des résultats
HOPFIELD		
Reconnaissance par caractéristiques	- faisable - liste de candidats	- non-pratique - temps de réponse
RESEAU D'ORDRE SUPERIEUR		
Reconnaissance sur bitmaps	- espace mém. const. - temps de réponse; - temps d'apprentiss.	- explosion combinatoire du nombre de connexions.

METHODE CLASSIQUE

Reconnaissance par caractéristiques en base dix	- temps de réponse si échelle = 1; - mémorisation du modèle.	- temps de réponse très longue si l'échelle ≠ 1.

Dans ce travail nous voulons reconnaître un formulaire introduit parmi un nombre de formulaires que le système a emmagasiné. Cette reconnaissance doit s'effectuer à l'aide des lignes horizontales et verticales du formulaire et malgré les transformations éventuelles.

Nous avons analysé de façon pratique quelques réseaux neuronaux afin de percevoir leur capacité de reconnaissance.

Deux types de réseaux fréquemment repris dans les travaux sur les réseaux neuronaux sont le Perceptron et le réseau Hopfield. Dans ce mémoire il est démontré qu'aucun n'est satisfaisant.

Le réseau neuronal d'ordre supérieur à une seule couche permet un apprentissage facile et l'introduction d'invariance. Nos tests, limités à la translation et à la modification d'échelle, se sont avérés fructueux, mais les bitmaps utilisés sont trop petits pour assurer le succès requis dans un produit fini. La robustesse est garantie par les propriétés émanantes des réseaux neuronaux. Mais dans un environnement réel le réseau neuronal, de par sa nature, nécessite un support parallèle pour vraiment prouver ces capacités.

La méthode classique satisfait quant à la reconnaissance de formulaires qui ont subi des transformations, et quant à la robustesse. Malgré que la version implémentée n'est pas optimalisée, les temps de réponse sont acceptables si le formulaire existe dans la base de données.

En guise de conclusion générale, nous pouvons avancer que ni la méthode classique ni le réseau neuronal n'est idéal. Mais j'affirme que les systèmes de vision, dont la reconnaissance de formulaires n'est qu'une infime partie, devront s'appuyer sur les deux méthodes précédentes.

VI.3. Recherche ultérieure.

Ce mémoire aimerait être un tremplin pour des recherches ultérieures. Quelques chemins de recherche ont été définitivement fermés. Mais ce qui est important, c'est qu'il démontre que certaines voies sont prometteuses. En voici quelques-unes:

Le réseau neuronal d'ordre trois a été brièvement abordé. Son potentiel pour la reconnaissance, indépendamment de la translation, de l'échelle et de la rotation est réel, vu le succès du réseau d'ordre deux présenté dans ce travail.

Passer de l'observation des résultats extérieurs au stade de l'introspection : essayer d'interpréter les poids des connexions. La méthode peut être empirique mais les résultats peuvent être pratiques (par exemple, retrouver les valeurs des paramètres de transformation) sans pour autant s'adonner à l'analyse théorique de tout le fonctionnement. Même si cette compréhension théorique doit être recherchée.

Une autre voie à suivre est celle de la reconnaissance des formulaires par des caractéristiques plus appropriées aux réseaux neuronaux. Une des propriétés émergentes des réseaux neuronaux est la généralisation. Un réseau peut apprendre des patterns à partir d'exemples. En apportant des exemples bruités et différents, le réseau extrait les caractéristiques importantes des patterns. La question est donc : quelles sont les caractéristiques qu'un réseau extrait d'un formulaire et pourquoi? Et si on lui apprend des frames?

Il reste encore beaucoup d'autres idées et de chemins inexplorés et le potentiel du réseau neuronal est encore loin d'être épuisé!

RESEAUX NEURONAUX ET SCIENCES DE LA COGNITION

De nos jours, les applications de pattern matching ont fait leurs preuves, surtout dans le domaine visuel et auditif, et même dans le domaine de l'équilibre. Celles-ci sont les résultats de la première étape de vie des réseaux neuronaux dans les domaines scientifiques : la recherche et les "toy worlds". Le pas suivant, l'application industrielle, est en pleine expansion. Les réseaux vont faire preuve de plus de qualités encore que celles dont nous sommes conscients aujourd'hui. En même temps, les applications vont faire ressortir les limites. Peu à peu, le domaine des réseaux neuronaux va prendre sa place dans les sciences. Son utilisation deviendra aussi commune que les autres applications, car il répond à des besoins insatisfaits jusqu'à présent. Mais nous n'y sommes pas encore.

Il est déjà possible de "ressentir intuitivement" la place que les réseaux neuronaux vont prendre. Si les systèmes experts trouvent leur force dans le raisonnement séquentiel et sa justification, les réseaux, eux, favorisent l'émergence de qualités difficilement analysables par le système lui-même.

Les deux rappellent le fonctionnement cérébral. Intuitivement nous savons attribuer les processus subconscients aux réseaux neuronaux, et le raisonnement aux systèmes experts. Or, quelle est l'interaction entre les deux? Le raisonnement fait partie de l'ensemble des processus conscients, qui supervisent également l'apprentissage. Le cerveau ne contient pas de machine "von Neumann". Alors comment les processus conscients sont-ils implémentés? (Un réseau neuronal superposé à un autre?) Finalement un jour, peut-être, nous aborderons le pas ultime, la

compréhension de la conscience d'être conscient. La conscience qui rend l'homme unique parmi les êtres terrestres.

Voici les questions auxquelles des réseaux neuronaux peuvent apporter des réponses partielles. Mais pour trouver ces réponses, il faut d'abord connaître les réseaux et leurs potentiels. Nous ne sommes qu'au début de cette route.

La beauté du domaine neuronal se trouve dans l'apport interdisciplinaire. Les réseaux neuronaux formels se doivent de répondre aux contraintes posées notamment par la neuro-science, la psychologie et la théorie de la calculabilité, mais en même temps chacune de ces disciplines apporte ses connaissances. L'ère des sciences humaines considérées comme "agréable passe-temps, mais non comme science" est passée. La compréhension des processus cognitifs nécessite la coopération de tous.

ANNEXE A : GLOSSAIRE

BITMAP : matrice de pixels, normalement une image digitalisée.

FORMULAIRE : un document standard contenant de l'information préimprimé et des espaces blancs à remplir.

FRAME : les lignes extraites d'un formulaire.

HOPFIELD : un réseau récurrent qui implémente la satisfaction de contraintes.

NEURONE : entité formelle simulant le fonctionnement du neurone biologique.

PATTERN : ensemble de pixels sur un bitmap, avec les mêmes valeurs.

PERCEPTRON : réseau non-récurrent qui implémente un calcul de fonction.

PIXEL : (picture element) un point sur une image, à valeur 0 ou 1 dans notre cas.

RECONNAISSANCE : association entre un élément réel fourni au système de l'extérieure, et un élément de référence, aussi appelé modèle, stocké par le système. Un élément peut être un frame, un bitmap, ...

RESEAU NEURONAL : ensemble de neurones interconnectés.

RESEAU D'ORDRE SUPERIEUR : réseau dont les neurones d'entrée sont pris en tuples pour le calcul de la fonction d'activation.

TRANSFORMATION : une translation et/ou une rotation et/ou une modification d'échelle.

ANNEXE B : DETAIL ALGORITHMIQUE

B.1. RESEAU D'ORDRE SUPERIEUR

B.1.1. FICHIER D'ENTÊTE "HONN.H"

Ce fichier définit les constantes du programme (voir IV.2. et IV.3.)

```
#include <math.h>
#include <stdio.h>

#define BUFFSTRLEN 50
#define DATAPATH "D:\\PROJECT\\NN\\"
#define FALSE 0
#define MAXERROR 0.1
#define NINPUT 49
#define NOUTPUT 4
#define TRUE 1
```

B.1.2. FICHIER D'ENTÊTE "HONNVAR.H"

Dans ce fichier se trouvent les prototypes des structures de données.

```
#include "HONN.H"

extern  float activ0[NINPUT];
extern  float activ1[NOUTPUT];
extern  float trpat[NOUTPUT];
extern  float w2[NINPUT][NINPUT][NOUTPUT];
extern  int   link[NINPUT][NINPUT];

extern  int   MSG;
extern  int   learn;
extern  int   honn;
extern  float distance, seriedistance;
```

B.1.3. FICHIER DU PROGRAMME PRINCIPAL : "HONN.C"

Ce programme est l'implémentation de l'algorithme du IV.2.

```
#include "HONN.H"

float activ0[NINPUT];                    /* neuron activation of layer 0  */
float activ1[NOUTPUT];                 /* neuron activation of layer 1  */
float trpat[NOUTPUT];                   /* training pattern               */
float w2[NINPUT][NINPUT][NOUTPUT];      /* 2nd order weights              */
int   link[NINPUT][NINPUT];            /* weights with same value        */

int   MSG   = FALSE;                   /* debugging flag                 */
int   learn = FALSE;                    /* training session flag          */
int   honn  = TRUE;                    /* invariance flag                */
float seriedistance, distance;         /* error between patterns         */
                /* (serie = cycle)                 */

main(int argc, char *argv[])

{
  int   i,j,d,pttns,cycle;             /* i&j : origin   d(estination)  */
  int   nbrcycles;                     /* # of cycles of learn          */
  int   nbrpttns;                      /* number of patterns to process */
  char  ptname[6];                     /* pattern root name             */
  char  tempstring[11];

  /*** ARG LIST ***/
  if (argc < 5)
    { printf("usage: HONN -p <patternrootname>\n");
      printf("            -n <number of patterns>\n");
      printf("            -l <maximum number of learning cycles>\n");
      printf("           [-i] = disable invariance \n");
      printf("           [-d] = debug. \n");
      exit(0); };

  i=0;
  while (++i<argc)
   {      if (!strcmp(argv[i],"-p")) strcpy(ptname,argv[++i]);
     else if (!strcmp(argv[i],"-n")) nbrpttns = atoi(argv[++i]);
     else if (!strcmp(argv[i],"-l")) { nbrcycles = atoi(argv[++i]);
                                       learn = TRUE; }
     else if (!strcmp(argv[i],"-d")) MSG=TRUE;
     else if (!strcmp(argv[i],"-i")) honn=FALSE;
     else { printf("HONN : unrecognized argument.\n"); exit(0); };
   };

  /*** MESSAGE ***/
  if (MSG) { printf("HONN : executing main.\n"); };

  /*** INITIALISATIONS ***/
  for (i=0; i<NINPUT; i++)
    for (j=0; j<NINPUT; j++)
      for (d=0; d<NOUTPUT; d++)
       w2[i][j][d]=((float)(rand()%1000))/1000.0;

  /*** EXECUTE PERCEPTRON ***/
  if (!learn)
    {
      ReadNetwork(ptname);
      for (pttns=1; pttns<=nbrpttns; pttns++)
       { /*** READ AN INPUT PATTERN ***/
         sprintf(tempstring,"%s%d.PAT",ptname,pttns);
         ReadPattern(activ0, tempstring);
         /*** CALCULATE THE RESULT PATTERN ***/
         CalcForward();
         ShowResult(activ1,NOUTPUT,FALSE,
                    "The result pattern is :");
         if (MSG) ShowNetwork(tempstring);
       };
```

```
    }
  else
    { sprintf(tempstring,"L%d.LKS",NINPUT);
      ReadLinks(tempstring);
      if (honn) CorrectLinks();
      for (cycle=1; cycle<=nbrcycles; cycle++)
      {
       seriedistance=0.0;
       for (pttns=1; pttns<=nbrpttns; pttns++)
       {
         /*** READ INPUT PATTERN ***/
         sprintf(tempstring,"%s%d.PAT",ptname,pttns);
         ReadPattern(activ0, tempstring);

         /*** READ TRAINING PATTERN ***/
         sprintf(tempstring,"%s%d.TRA",ptname,pttns);
         ReadPattern(trpat, tempstring);

         /*** CALCULATE FORWARD ***/
         CalcForward();

         /*** CALCULATE PATTERN ERROR ***/
         CorrectWeights();
         if (honn) CorrectLinks();
         CalcError();
         seriedistance += distance;


         /*** DEBUGGING AND VISUALISATION ***/
         if (MSG) ShowNetwork(tempstring);
       };
       printf("seriedistance for cycle %d : %f\n",cycle,seriedistance);
       if (seriedistance <= MAXERROR)
         break;  /* All patterns were learned */
       };

       /*** IF THE TRAINING WAS A SUCCESS, SAVE THE NETWORK ***/
       if (seriedistance <= MAXERROR)
       { WriteNetwork(ptname);
         printf("\nThe network learned the %d patterns in %d cycles.\n",
                nbrpttns, cycle);
       }
       else
       { printf("\nThe network did NOT learn the %d patterns\n", nbrpttns);
         printf("in the maximum allowed %d cycles.\n", cycle);
       };
    };

  /*** END ***/
  if (MSG) { printf("HONN : end of main.\n"); };
  return(0);
}.
```

B.1.4. FICHIER DES CALCULS : "HONNPROC.C"

Ce fichier contient les routines de calcul dont les formules se trouvent dans le les points II.6 et IV.2.

```
#include "HONNVAR.H"
```

```
/* ================================================================ */
void CalcForward()
{
  int   i,j,d;    /* d(estination node)     i & j = origin nodes */
  float dint;

  for (d=0; d<NOUTPUT; d++)
    {  activ1[d] = 0.0;
       /** CALCULATE  NET(d) **/
       for (i=0; i<NINPUT; i++)
        for (j=i; j<NINPUT; j++)
          activ1[d] += activ0[i] * activ0[j] * w2[i][j][d];
       /*** LOGISTIC F ***/
       activ1[d] = 1.0 / (1.0 + (float)exp((double)(-1.0 * activ1[d])));
    };
}

/* ================================================================ */
void CalcError()
{
  int   d;

  distance = 0.0;
  for (d=0; d<NOUTPUT; d++)
    distance += (trpat[d] - activ1[d]) * (trpat[d] - activ1[d]);
}

/* ========================== delta ================================ */
void CorrectWeights()
{
  int i,j,d;

   for (d=0; d<NOUTPUT; d++)
    { for (i=0; i<NINPUT; i++)
       for (j=i; j<NINPUT; j++)
         w2[i][j][d] += (trpat[d] - activ1[d]) * activ0[i] * activ0[j];
    };
}

/* ========================== avg ================================== */
void CorrectLinks()
{
  int i,j,d;                          /* i&j : origins  ; d(estination) */
  int stoplink, nextlink, ni, nj;
  float sum, nbrsum;

  for (d=0; d<NOUTPUT; d++)
    { /*** follow all linkrings ***/
      for (i=0; i<NINPUT; i++)
       for (j=i; j<NINPUT; j++)
         { if (link[i][j]>0)                /* negative links = "done" */
             { /*** make the sum ***/
               stoplink = i*NINPUT+j;
               sum = w2[i][j][d];
               nbrsum = 1.0;
               nextlink = link[i][j];
               while (nextlink != stoplink)
                 { ni=nextlink / NINPUT;
                   nj=nextlink % NINPUT;
                   sum += w2[ni][nj][d];
                   nbrsum+=1.0;
                   nextlink = link[ni][nj];
                 };
               /*** update the weights ***/
               w2[i][j][d] = sum/nbrsum;
               nextlink = link[i][j];
               while (nextlink != stoplink)
                 { ni=nextlink / NINPUT;
```

```
                nj=nextlink % NINPUT;
                w2[ni][nj][d] = sum/nbrsum;
                nextlink = link[ni][nj];
                link[ni][nj] = -link[ni][nj];    /* finished node */
              };
            };
        };
      /*** set links positive (= unused) ***/
      for (i=0; i<NINPUT; i++)
       for (j=i; j<NINPUT; j++)
         link[i][j] = abs(link[i][j]);
    };
}
```

B.1.5. FICHIER DES ENTRÉES/SORTIES : "HONNIO.C"

Les routines de ce fichier gèrent les entrées/sorties vers écran et mémoire auxiliaire.

```
#include "HONNVAR.H"

/* ================================================================== */
void ReadPattern(float *pat, char *infname)

{
  FILE *fopen(), *f_in;
  char buffstr[BUFFSTRLEN];
  int lcount, i;


  /*** MESSAGE ***/
  if (MSG)
    { printf("ReadPattern : reading file %s\n", infname); };

  /*** OPEN PAT FILE ***/
  sprintf(buffstr, "%s%s", DATAPATH, infname);
  f_in = fopen(buffstr, "r");
  if (f_in == NULL)
    { printf("ReadPattern : cannot open %s file.\n", buffstr);
      exit(1);};

  /*** READ PATTERN ***/
  fgets(buffstr,BUFFSTRLEN,f_in);
  sscanf(buffstr,"%d",&lcount);
  if (lcount>NINPUT)
    { printf("ReadPattern : too many nodes in %s.\n", buffstr);
      exit(1);};

  for (i=0; i<lcount; i++)
    { fgets(buffstr,BUFFSTRLEN,f_in);
      sscanf(buffstr,"%f",&pat[i]);
    };

  fclose(f_in);
}

/* ================================================================== */
void ReadLinks(char *infname)

{
```

```
  FILE *fopen(), *f_in;
  char buffstr[BUFFSTRLEN];
  int i,j;


  /*** MESSAGE ***/
  if (MSG)
    { printf("ReadLinks : reading file %s\n", infname); };

  /*** OPEN PAT FILE ***/
  sprintf(buffstr, "%s%s", DATAPATH, infname);
  f_in = fopen(buffstr, "r");
  if (f_in == NULL)
    { printf("ReadLinks : cannot open %s file.\n", buffstr);
      exit(1);};

  /*** READ LINKS ***/
  for (i=0; i<NINPUT; i++)
   for (j=i; j<NINPUT; j++)
    { fgets(buffstr,BUFFSTRLEN,f_in);
      sscanf(buffstr,"%d",&link[i][j]);
    };

  fclose(f_in);
}


/* ================================================================== */
void ShowResult(float *pat, int lgth, int pattern, char *title)

{
  double dtmp;
  int    lin, col, l, c;

  /*** PRINT TITLE ***/
  printf("\n%s\n",title);

  /*** SQUARE PATTERN??? ***/
  lin = lgth;
  col = 1;
  if (modf( sqrt((double)lgth), &dtmp) == 0)
    { lin = (int)sqrt((double)lgth);
      col = lin; };

  /*** PRINT PATTERN ***/
  for (l=0; l<lin; l++)
    { for (c=0; c<col; c++)
       { if (pattern)
           { if (pat[l*lin+c]==1.0)
                printf("²²");
             else printf("__");
           }
         else
           { printf("%4.2f ",pat[l*lin+c]); };
       };
      printf("\n");
    };
}

/* ================================================================== */
void WriteNetwork(char *rootname)

{ FILE *fopen(), *fout;
  char buffstr[BUFFSTRLEN];
  int i,j,d;

  /*** OPEN FILE ***/
  sprintf(buffstr,"%s%s.WTS",DATAPATH,rootname);
```

```
  fout = fopen(buffstr,"w");
  if (fout == NULL)
    { printf("HONN : cannot write file %s%s.WTS\n", DATAPATH, rootname);
      exit(0); };

  /*** WRITE VALUES ***/
  /** activation values in nodes **/
  fprintf(fout,"%04d %04d\n", NINPUT, NOUTPUT);
  for (i=0; i<NINPUT; i++)
    fprintf(fout,"%05.2f\n",activ0[i]);
  for (d=0; d<NOUTPUT; d++)
    fprintf(fout,"%05.2f\n",activ1[d]);

  /** weigths **/
  for (i=0; i<NINPUT; i++)              /* origin nodes      */
    for (j=0; j<NINPUT; j++)            /* origin nodes      */
      for (d=0; d<NOUTPUT; d++)         /* destination nodes */
       fprintf(fout,"%09.5f\n", w2[i][j][d]);

  /*** CLOSE ***/
  fclose(fout);
}


/* ================================================================== */
void ReadNetwork(char *rootname)

{ FILE *fopen(), *fin;
  char buffstr[BUFFSTRLEN];
  int i,j,d;

  /*** OPEN FILE ***/
  sprintf(buffstr,"%s%s.WTS",DATAPATH,rootname);
  fin = fopen(buffstr,"r");
  if (fin == NULL)
    { printf("HONN : cannot read file %s%s.WTS\n", DATAPATH, rootname);
      exit(0); };

  /*** READ VALUES ***/
  /** activation values in nodes **/
  fgets(buffstr,BUFFSTRLEN,fin);
  sscanf(buffstr,"%d %d",&i, &j);
  if ((i!=NINPUT) || (j!=NOUTPUT))
    { printf("HONN : wrong network dimensions reading file %s%s.WTS",
            DATAPATH, rootname);
      exit(0); };

  for (i=0; i<NINPUT; i++)
    { fgets(buffstr,BUFFSTRLEN,fin);
      sscanf(buffstr,"%f",&activ0[i]);
    };
  for (d=0; d<NOUTPUT; d++)
    { fgets(buffstr,BUFFSTRLEN,fin);
      sscanf(buffstr,"%f",&activ1[d]);
    };

  /** weigths **/
  for (i=0; i<NINPUT; i++)
    for (j=0; j<NINPUT; j++)
      for (d=0; d<NOUTPUT; d++)
       { fgets(buffstr,BUFFSTRLEN,fin);
         sscanf(buffstr,"%f", &w2[i][j][d]);
       };

  /*** CLOSE ***/
  fclose(fin);
}
```

```
/* ================================================================== */
void ShowNetwork(char *namestr)

{ int i,j,d;
  char tmpstr[15];

  /*** title ***/
  printf("\n=========================\n%s\n\n",namestr);
  /*** input layer : node values ***/
 ShowResult(&activ0[0],NINPUT,FALSE, "Layer 0");
 /*** sequent layer : weights and node values ***/
  printf("Layer 1\n");
  for (i=0; i<NINPUT; i++)
    for (j=0; j<NINPUT; j++)
      ShowResult(&w2[i][j][0], NOUTPUT, FALSE, "Weights");
  ShowResult(&activ1[0],NOUTPUT,FALSE,"Node values");
  /*** error indication ***/
   printf("\nThe total error for this serie is as yet: %7.2f\n",
         seriedistance);
}
```

B.2. CREATION DE LA MATRICE DES LIENS.

Ce programme crée un fichier qui contient une matrice supérieure contenant tous les liens des connexions d'ordre 2 qui doivent avoir la même valeur.

```
#include <stdio.h>
#include <values.h>

#define DATAPATH "D:\\PROJECT\\NN\\"
#define ROWS 3
#define COLS 3
#define NODES 9

main()
{
  int x1, y1, x2, y2;
  int n1, n2, pn1, pn2, nn1, nn2;          /*  n(ext)n   p(rev)n */
  float slope[NODES][NODES];
  int   link[NODES][NODES];
  FILE *fopen(), *fout;
  char buffstr[50];

  /*** init slopes ***/
  for (n1=0; n1<NODES; n1++)
    for (n2=0; n2<NODES; n2++)
      slope[n1][n2]=0.0;

  /*** calc slopes ***/
  for (n1=0; n1<NODES; n1++)
  { x1=n1%COLS; y1=n1/COLS;
    for (n2=n1+1; n2<NODES; n2++)
    { x2=n2%COLS; y2=n2/COLS;
      if ((x2-x1)==0)
       slope[n1][n2]=MAXFLOAT;
      else
       slope[n1][n2]=((float)(y2-y1))/((float)(x2-x1));
    };
  };
```

```
  /*** init links ***/
  for (n1=0; n1<NODES; n1++)
    for (n2=0; n2<NODES; n2++)
      link[n1][n2]= -1;

  /*** link diagonal links ***/
  for (n1=0; n1<NODES-1; n1++)
    link[n1][n1]=(n1+1)*NODES+(n1+1);
  link[NODES-1][NODES-1]=0;

  /*** link links ***/
  for (n1=0; n1<NODES; n1++)
    for (n2=n1+1; n2<NODES; n2++)
      if (link[n1][n2] < 0)
       { link[n1][n2]=n1*NODES+n2;
         pn1=n1; pn2=n2;
         for (nn1=n1; nn1<NODES; nn1++)
           for (nn2=nn1+1; nn2<NODES; nn2++)
             if (slope[n1][n2]==slope[nn1][nn2])
              { link[pn1][pn2]=nn1*NODES+nn2;
                link[nn1][nn2]=n1*NODES+n2;
                pn1=nn1; pn2=nn2;
              };
       };


  /*** save links ***/
  /*** OPEN FILE ***/
  sprintf(buffstr,"%sL%d.LKS",DATAPATH,NODES);
  fout = fopen(buffstr,"w");
  if (fout == NULL)
    { printf("Cannot write file %sL%d.LKS\n", DATAPATH,NODES);
      exit(0); }
  else
    printf("writing file %sL%d.LKS\n", DATAPATH,NODES);
  /*** WRITE VALUES ***/
  for (n1=0; n1<NODES; n1++)
    for (n2=n1; n2<NODES; n2++)
      fprintf(fout,"%05d\n",link[n1][n2]);
  /*** CLOSE ***/
  fclose(fout);

return(0);
}
```

ANNEXE C : BIBLIOGRAPHIE

[Cohen, 1983]
"Absolute stability of global pattern formation and parallel memory storage by competitive neural networks."
M.A. Cohen & S.G.Grossberg;
{IEEE trans. on systems, man and cybernetics, vol13, pp815-826,1983}

[Grimson, 1990]
"Hough transform and object recognition."
Grimson & Huttenlocher;
{IEEE trans. on PAMI, vol12, pp255-274, 1990}

[Hecht, 1987]
"Counterpropagation networks."
R. Hecht-Nielsen;
{Proceedings of the IEEE first international conference on neural networks, vol2, pp 19-32, 1987}

[Honet, 1990]
"Les réseaux de neurones formels et leurs applications en reconnaissance des formes."
L. Honet;
{Mémoire de licence en sciences physiques, FUNDP, Namur, 1990}

[Hopfield, 198]
"Neural networks and physical systems with emergent collective computational abilities."
J.J. Hopfield;
{Proceedings of the National Academy of Sciences, vol79, pp2554-2558, 1982}

[Kirkpatrick, 1983]
"Optimization by simulated annealing."
S. Kirkpatrick, C.D.jr. Gelatt & M.P. Vecchi;
{Science, vol220, pp671-680, 1983}

[Li, 1989]
"Object recognition based on graph matching implemented by a Hopfield style network."
W. Li & N.M. Nasrabadi;
{IEEE INNS : International Joint conference on Neural Networks, vol2, pp287-290, 1989}

[McCulloch, 1943]
"A logical calculus of the ideas immanent in nervous activity."

W.S. McCulloch & W. Pitts;
{Bulletin of Mathematical Biophysics, vol 5, pp115-133, 1943}

[Minsky, 1969]
"Perceptrons."
M. Minsky & S. Papert;
{The MIT Press, 1969}

[Nivelles, 1990]
"Les réseaux neuronaux : Approche théorique et applications."
P. Nivelles & V. Thiry;
{Mémoire de licence en informatique, FUNDP, Namur, 1990}

[Parvin, 1989]
"A constraint satisfaction network for matching."
B. Parvin & G. Medioni;
{IEEE INNS : International joint conference on neural networks, vol2, pp281-286, 1989}

[Philips-1, 1990]
"Form recognition."
K. Bekaert;
{Nat Lab Technical Note No. 060/91}

[Philips-2, 1990]
"Extraction of filled in information from forms."
S. Mendez-Porcel & P. Olivié;
{Nat Lab Technical Note No. 033/91}

[Pitts, 1947]
"How we know universals."
W. Pitts & W.W. McCulloch;
{Bulletin of mathematical biophysics, vol9, pp127-147, 1947}

[Reid, 1990]
"Rapid training of higher-order neural networks for invariant pattern recognition."
M.B. Reid, L. Spirkovska & E. Ochoa;
{IEEE INNS : International joint conference on neural networks, vol2, pp689-692, 1990}

[Rosenblatt, 1962]
"Principles of neurodynamics."
F. Rosenblatt;
{Spartan, 1962}

[Rumelhart-1, 1986]
"Parallel Distributed Processing : Volume 1 : Foundations."
D.E. Rumelhart, J.L. McClelland & PDP Research Group;
{The MIT Press, 1986}

[Rumelhart-2, 1986]
"Parallel Distributed Processing : Volume 2 : ."

J.L. McClelland, D.E. Rumelhart & PDP Research Group;
{The MIT Press, 1986}

[Rumelhart-3, 1989]
"Explorations in Parallel Distributed Processing."
J.L. McClelland, D.E. Rumelhart;
{The MIT Press, 1989}

[Tresp, 1990]
"Invariant object recognition by inexact subgraph matching with applications in industrial part recognition."
V. Tresp;
{Proceedings of the international neural networks conference, Paris, 1990}

[Wasserman, 1989]
"Neural computing. Theory and practice."
P.D. Wasserman;
{Van Nostrand Reinhold, 1989}

[Werbos, 1974]
"Beyond regression; New T."
P. Werbos;
{Ph.D. Thesis, Harvard University, 1974}